

FOR FURTHER TRAN

ARPA ORDER NO. 2223

AD A 055516

12

J

ISI/RR-77-66

November 1977



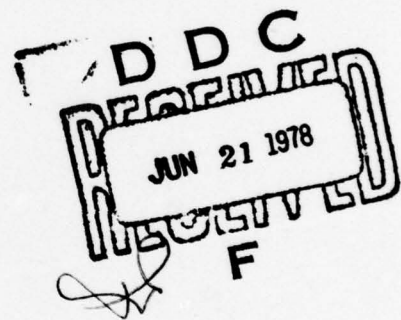
Mark S. Moriconi

University of Texas at Austin
and USC/Information Sciences Institute

A System for Incrementally Designing and Verifying Programs

(Appendix) Volume 2

AD NO. _____
DDC FILE COPY



This document has been approved
for public release and sale; its
distribution is unlimited.

INFORMATION SCIENCES INSTITUTE

UNIVERSITY OF SOUTHERN CALIFORNIA



4676 Admiralty Way/ Marina del Rey/ California 90291

(213) 822-1511

78 06 19 059

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 14 ISI/RR-77-66	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A System for Incrementally Designing and Verifying Programs, Vol. I		5. TYPE OF REPORT & PERIOD COVERED Research rept.
6. AUTHOR(s) 14 Mark S. Moriconi		7. CONTRACT OR GRANT NUMBER(s) 15 DAHC 15-72-C-0308, MC574-12866 ARPA Order 2223
9. PERFORMING ORGANIZATION NAME AND ADDRESS USC/Information Sciences Institute 4676 Admiralty Way Marina del Rey, CA 90291		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE 11 January 1978
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		13. NUMBER OF PAGES 66 12 69p.
		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) This report is approved for public release and sale; distribution is unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) asynchronous, data abstraction, incremental, interactive dialogue, interactive system, network, program design, program verification, security, specifications		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This appendix contains a transcript of a session with SID in which a simple message switching network that allows secure, asynchronous message transfer among a fixed number of users is incrementally developed. A		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-660178 06 19 059 UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

447 952

act



Mark S. Moriconi

University of Texas at Austin
and USC/Information Sciences Institute

A System for Incrementally Designing and Verifying Programs
(Appendix) Volume 2

INFORMATION SCIENCES INSTITUTE

UNIVERSITY OF SOUTHERN CALIFORNIA



4676 Admiralty Way/ Marina del Rey/ California 90291
(213) 822-1511

THIS RESEARCH IS SUPPORTED BY THE ADVANCED RESEARCH PROJECTS AGENCY UNDER CONTRACT NO. DAHC15 72 C 0308, ARPA ORDER NO. 2223.

VIEWS AND CONCLUSIONS CONTAINED IN THIS STUDY ARE THE AUTHOR'S AND SHOULD NOT BE INTERPRETED AS REPRESENTING THE OFFICIAL OPINION OR POLICY OF ARPA, THE U.S. GOVERNMENT OR ANY OTHER PERSON OR AGENCY CONNECTED WITH THEM.

THIS DOCUMENT APPROVED FOR PUBLIC RELEASE AND SALE: DISTRIBUTION IS UNLIMITED.

APPENDIX

MESSAGE SWITCHING NETWORK

This appendix contains a transcript of a session with SID in which a simple message switching network is incrementally designed and verified using a top-down strategy. Interleaved with the transcript are annotations explaining the development. This commentary focuses on how SID is used, rather than on SID's internal operation. An overview of what SID does in various situations is given in the example session in Section 1.2, and then elaborated throughout the text.

The network, which allows secure, asynchronous message transfer among a fixed number of users, is developed in several key stages:

- **User/network interface.** The network switches messages among a fixed number of user processes. These processes operate concurrently with the network and, except for their interface with the network, their operation is unspecified. Each user process communicates with the network through a port, as shown in Fig. A-1. Each port consists of an input buffer and an output buffer and has an associated security classification.
- **Network specification.** The network specification states that the mail received by user j is a subsequence of the secure mail intended for j . This requires that messages for each port arrive in the same order they were sent. The subsequence relation allows messages to be dropped due to unrecoverable transmission failures or security violations. This property is formalized using a hierarchically-structured collection of specifications.
- **Network implementation and verification.** The network is implemented as a set of concurrently running message switching processes, with each switcher accepting messages from only one user input buffer and routing messages to any output buffer. This configuration is shown in Fig. A-2. Also, properties about the switchers are stated and used in the verification of the top-level network program. These specifications say that for every pair of ports i, j , the mail actually sent from source i to destination j is a subsequence of the secure mail from i intended for j .
- **Switcher implementation and verification.** The switchers receive messages, determine their destination and check security, then send the message if nothing is wrong.

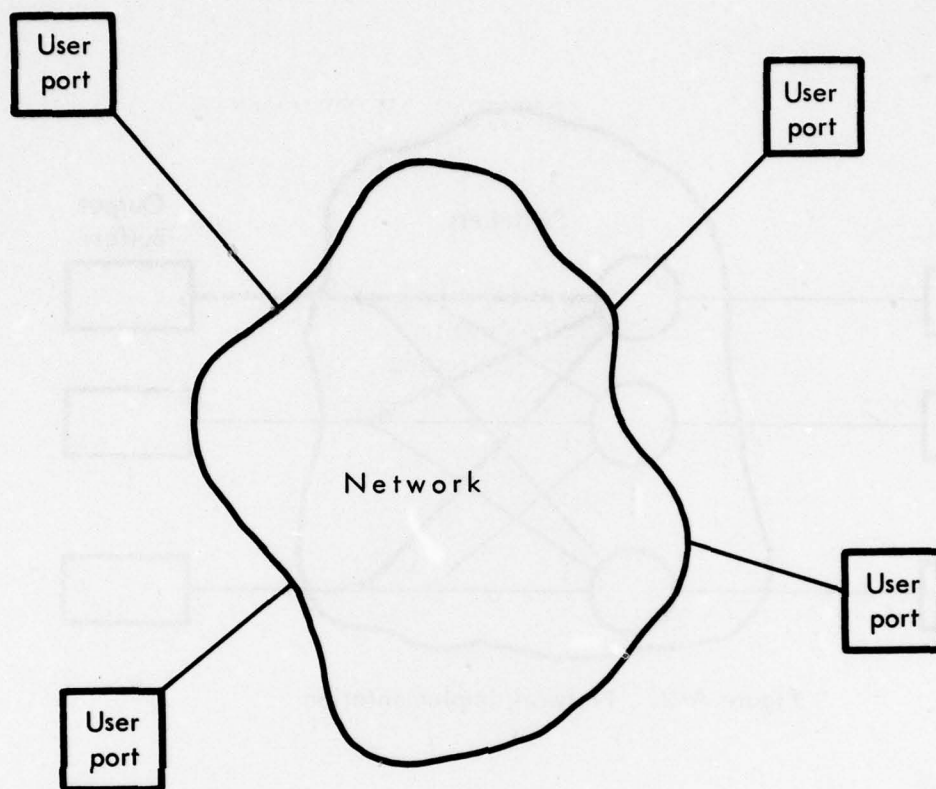


Figure A-1. User/network interface

- ***Message implementation and verification.*** Previously, messages were manipulated without knowledge of their internal structure. The message abstraction is now implemented as a record of three fields -- a source identification, a destination identification, and the message text. The functions which manipulate the internal structure of messages are also implemented and verified, completing the development.

Until the last stage, several partially-defined programs and data definitions are entered and refined. Annotations relate these intermediate stages of development to the above outline.

The transcript can be read at several levels of detail. An overview is obtained by ignoring specifics of the code, specifications, VC generation trace, and proofs. Focus should be

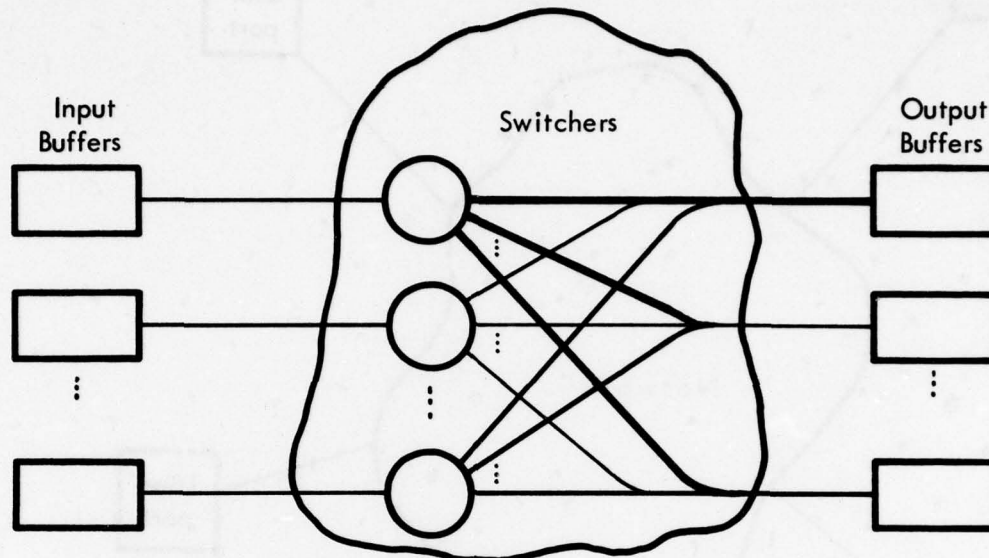


Figure A-2. Network implementation

on the kinds of things being done and their relation to the development scenario given above. After an initial reading, the transcript can be studied at increasingly finer levels of detail.

Annotations and status summaries aid in keeping track of progress and context. Annotations, in addition to providing general commentary, point out important transitions in the development. Status summaries are frequently printed by SID to describe the current state of the entire development.

As in Section 1.2, lines typed by a person are underlined to distinguish them from the computer's response. The text is taken verbatim from an actual session, and is all inclusive except where explicitly marked. Italicized lines are comments describing the flow of the development. The final version of the network, its specifications, and all used lemmas are displayed at the end of the session.

Exec-> ?

Commands are: Add, Delete, EDit, EXplain, Help, PRint, PROve,
Restore, SAve, SUGgest, Translate, Vcs, ?E, ?, ??.

Exec-> Suggest

The user asks SID to tell him how to get started. The entire development is guided by SID's suggestions.

Suggest one of:

<esc> TRANSLATE (from file)

RESTORE (problem from file)

-> 1

Enter file name-> NetInt.Net

```

process Network(var aUserPort:UserPortArray;
                Security:SecurityArray) = pending;

type UserPortArray = array(UserId) of UserPort;

type UserId = integer[1..NUsers];

const NUsers = 100;

type UserPort =
  record ( SysIn:MsgInBuf;
           SysOut:MsgOutBuf
         );

type MsgInBuf = buffer (1) of Msg;

type MsgOutBuf = buffer (NUsers) of Msg;

type Msg = pending;

type SecurityArray = array([1..NUsers]) of SecurityClass;

type SecurityClass =
  (unclassified,confidential,secret,topsecret);

```

No syntax errors detected.
No semantic errors detected.

ACCESSION for	
NTIS	White Section <input checked="" type="checkbox"/>
DDC	Buff Section <input type="checkbox"/>
UNANNOUNCED	<input type="checkbox"/>
JUSTIFICATION	<input type="checkbox"/>
BY _____	
DISTRIBUTION/AVAILABILITY CODES	
Dist	SPECIAL
A	

The user/network interface is established by precisely defining the parameters to the network. The implementation of messages is postponed.

Suggest fully defining NETWORK-> Print Status All

The current design and verification status is:

Waiting for pending body to be filled in: NETWORK

Constants/Types: MSG, MSGINBUF, MSGOUTBUF, NUSERS, SECURITYARRAY, SECURITYCLASS, USERID, USERPORT, USERPORTARRAY

Suggest fully defining NETWORK-> Edit

.
.
.

Exec-> Translate NetSpc.Net

```
process Network(var aUserPort:UserPortArray;
                Security:SecurityArray) =
begin
  block all j:userid,
    issubmerge(auserport(j).sysout.outlo,
              allsecuremailfor(auserport,j,security));
  pending
end;

function IsSubMerge(s:MsgSeq;
                  a:MsgSeqArray) : Boolean =
begin
  exit (assume IsSubMerge(s,a)
        iff (some x:MsgSeq, IsMerge(x,ArrayToBag(a,1,Nusers))
              and s sub x));
end;

type MsgSeq = sequence of Msg;

type MsgSeqArray = array (Userid) of MsgSeq;

function ArrayToBag(a: MsgSeqArray; i,j: Userid): BagMsgSeq =
begin end;
```

```

type BagMsgSeq = Bag of MsgSeq;

function AllSecureMailFor(A:UserPortArray;
                        j:integer;
                        s:SecurityArray): MsgSeqArray =
begin
  exit (assume all i:Userid,
        AllSecureMailFor(a,j,s)(i)
        = SecureMail(s(i),s(j),Mail(a(i).sysin.infrom,i,j)));
end;

function SecureMail(s1,s2:SecurityClass;
                  ms:MsgSeq):MsgSeq =
begin end;

function Mail(ms:MsgSeq;i,j:Userid):MsgSeq =
begin end;

```

The network specification uses several pre-defined constructs, some of which require explanation. The "infrom" and "outto" histories record transactions on buffers. In the specification of Network, auserport(j).sysout.outto is the sequence of all messages sent out to buffer auserport(j).sysout by the network. The infrom history refers to messages received by the network. The function "ismerge" determines if there exists a merge of a bag of sequences which is equal to a given sequence. Functions having only their headers defined (e.g., SecureMail) are used for type-checking purposes.

No syntax errors detected.
No semantic errors detected.

Exec-> Print Status All

The current design and verification status is:

Waiting for pending body to be filled in: NETWORK

For specifications only: ARRAYTOBAG, ALLSECUREMAILFOR, ISSUBMERGE, MAIL,
SECUREMAIL

Constants/Types: BAGMSGSEQ, MSG, MSGSEQ, MSGINBUF, MSGOUTBUF,
MSGSEQARRAY, NUSERS, SECURITYARRAY, SECURITYCLASS, USERID,
USERPORT, USERPORTARRAY

Exec-> Suggest

Suggest fully defining NETWORK-> Edit

Exec-> Translate NetWrk.Net

```

process Network(var aUserPort:UserPortArray;
                Security:SecurityArray) =
begin
block all j:userid,
    issubmerge(auserport(j).sysout.outto,
                allsecuremailfor(auserport,j,security));
cobegin
    Switcher(i,aUserPort,Security)
    each i : [1..NUsers];
end;
end;

process Switcher(i:UserId; var aports:UserPortArray;
                Security:SecurityArray) =
begin
entry i in [1..NUsers];
block all j:userid,
    aports(j).sysout.outto sub
        securemail(security(i),security(j),
                    mail(aports(i).sysin.infrom,i,j))
    and
        ( j ne i -> aports(j).sysin.infrom = MsgSeq() );
pending
end;

function Infrom!\ArrayToBag (a:MsgBufArray; i,j:UserId) :
    BagMsgSeq = begin end;

function Outto!\ArrayToBag (a:MsgBufArray; i,j:UserId) :
    BagMsgSeq = begin end;

type MsgBufArray = array (UserId) of MsgOutBuf;

```

No syntax errors detected.

No semantic errors detected.

The pending in the network program is replaced by executable code. The switcher processes are set into concurrent execution by the cobegin statement. The specifications of Switcher are given for use in the proof of Network, but the implementation of the actual switching algorithm is postponed.

Exec-> Print Status All

The current design and verification status is:

Waiting for VC generation: NETWORK

Waiting for pending body to be filled in: SWITCHER

For specifications only: ARRAYTOBAG, ALLSECUREMAILFOR, ISSUBMERGE,
INFROM\ARRAYTOBAG, MAIL, OUTTO\ARRAYTOBAG, SECUREMAIL

Constants/Types: BAGMSGSEQ, MSG, MSGSEQ, MSGINBUF, MSGOUTBUF,
MSGBUFARRAY, MSGSEQARRAY, NUSERS, SECURITYARRAY, SECURITYCLASS,
USERID, USERPORT, USERPORTARRAY

As reflected in this status summary, the top-level network program is ready to be verified. VCs are generated and then proved.

Exec-> Suggest

Suggest generating VCs for NETWORK-> §

Generating VCs for PROCESS NETWORK

Found 1-ST path

SID traces the generation of the single VC for Network below.

Beginning new path...

AUSERPORT' := AUSERPORT

Assume Initial Buffer Histories are null

all [i] : INTEGER [1..100], AUSERPORT[i].SYSIN.INFROM = MSGSEQ()
 and AUSERPORT[i].SYSIN.OUTTO = MSGSEQ()
 and AUSERPORT[i].SYSOUT.INFROM = MSGSEQ()
 and AUSERPORT[i].SYSOUT.OUTTO = MSGSEQ()

Assume (unit entry condition)

TRUE

Entering cobegin . . .

Assume local buffer histories only changed by subprocesses

```

all I#3 : INTEGER [1..100],      AUSERPORT#1[I#3].SYSIN.INFROM
    = CO\AUSERPORT[I#3].SYSIN.INFROM
    @ AUSERPORT[I#3].SYSIN.INFROM
and AUSERPORT#1[I#3].SYSIN.OUTTO
    = CO\AUSERPORT[I#3].SYSIN.OUTTO
    @ AUSERPORT[I#3].SYSIN.OUTTO
and AUSERPORT#1[I#3].SYSOUT.INFROM
    = CO\AUSERPORT[I#3].SYSOUT.INFROM
    @ AUSERPORT[I#3].SYSOUT.INFROM
and AUSERPORT#1[I#3].SYSOUT.OUTTO
    = CO\AUSERPORT[I#3].SYSOUT.OUTTO
    @ AUSERPORT[I#3].SYSOUT.OUTTO

```

Assume no buffer actions of subprocesses lost

```

all I#4 : INTEGER [1..100],
    ISMERGE(CO\AUSERPORT[I#4].SYSIN.INFROM,
        INFROM\ARRAYTOBAG(AUSERPORT#1[I#4].SYSIN.PAR, 1, NUSERS))
and ISMERGE(CO\AUSERPORT[I#4].SYSIN.OUTTO,
    OUTTO\ARRAYTOBAG(AUSERPORT#1[I#4].SYSIN.PAR, 1, NUSERS))
and TRUE
and ISMERGE(CO\AUSERPORT[I#4].SYSOUT.INFROM,
    INFROM\ARRAYTOBAG(AUSERPORT#1[I#4].SYSOUT.PAR, 1, NUSERS))
and ISMERGE(CO\AUSERPORT[I#4].SYSOUT.OUTTO,
    OUTTO\ARRAYTOBAG(AUSERPORT#1[I#4].SYSOUT.PAR, 1, NUSERS))
and TRUE

```

Assume system blocks.

Assume blockage conditions for subprocesses.

Assume (blockage condition for subprocess SWITCHER)

```

all I#2 : [1..NUSERS],
    all J : USERID,
        AUSERPORT#1[J].SYSOUT.PAR[I#2].OUTTO
    sub SECUREMAIL(SEcurity[I#2], SEcurity[J],
        MAIL(AUSERPORT#1[I#2].SYSIN.PAR[I#2].INFROM,
            I#2, J))
    and (J ne I#2 -> AUSERPORT#1[J].SYSIN.PAR[I#2].INFROM = MSGSEQ())

```

Blockage assertion is

```
all J : USERID, ISSUBMERGE(AUSERPORT[J].SYSOUT.OUTTO,
                           ALLSECUREMAILFOR(AUSERPORT, J, SECURITY))
```

Must verify (process blockage) condition

Verification condition NETWORK#1

```
H1: all I#1 : INTEGER [1..100],   AUSERPORT[I#1].SYSIN.INFROM = MSGSEQ()
                                   and AUSERPORT[I#1].SYSIN.OUTTO = MSGSEQ()
                                   and AUSERPORT[I#1].SYSOUT.INFROM = MSGSEQ()
                                   and AUSERPORT[I#1].SYSOUT.OUTTO = MSGSEQ()
```

```
H2: all I#2 : [1..NUSERS],
    all J : USERID,
      ( I#2 ne J
      -> AUSERPORT#1[J].SYSIN.PAR[I#2].INFROM = MSGSEQ())
    and AUSERPORT#1[J].SYSOUT.PAR[I#2].OUTTO
      sub SECUREMAIL(SECURITY[I#2], SECURITY[J],
                    MAIL(AUSERPORT#1[I#2].SYSIN.PAR[I#2].INFROM,
                        I#2, J))
```

```
H3: all I#3 : INTEGER [1..100],
    CO\AUSERPORT[I#3].SYSIN.INFROM @ AUSERPORT[I#3].SYSIN.INFROM
    = AUSERPORT#1[I#3].SYSIN.INFROM
    and CO\AUSERPORT[I#3].SYSIN.OUTTO @ AUSERPORT[I#3].SYSIN.OUTTO
    = AUSERPORT#1[I#3].SYSIN.OUTTO
    and CO\AUSERPORT[I#3].SYSOUT.INFROM @ AUSERPORT[I#3].SYSOUT.INFROM
    = AUSERPORT#1[I#3].SYSOUT.INFROM
    and CO\AUSERPORT[I#3].SYSOUT.OUTTO @ AUSERPORT[I#3].SYSOUT.OUTTO
    = AUSERPORT#1[I#3].SYSOUT.OUTTO
```

```
H4: all I#4 : INTEGER [1..100],
    ISMERGE(CO\AUSERPORT[I#4].SYSIN.INFROM,
            INFROM\ARRAYTOBAG(AUSERPORT#1[I#4].SYSIN.PAR, 1, NUSERS))
    and ISMERGE(CO\AUSERPORT[I#4].SYSIN.OUTTO,
            OUTTO\ARRAYTOBAG(AUSERPORT#1[I#4].SYSIN.PAR, 1, NUSERS))
    and ISMERGE(CO\AUSERPORT[I#4].SYSOUT.INFROM,
            INFROM\ARRAYTOBAG(AUSERPORT#1[I#4].SYSOUT.PAR, 1, NUSERS))
    and ISMERGE(CO\AUSERPORT[I#4].SYSOUT.OUTTO,
            OUTTO\ARRAYTOBAG(AUSERPORT#1[I#4].SYSOUT.PAR, 1, NUSERS))
```

-->

```
C1: all J : USERID, ISSUBMERGE(AUSERPORT#1[J].SYSOUT.OUTTO,
                                ALLSECUREMAILFOR(AUSERPORT#1, J, SECURITY))
```

End of path

Suggest proving VC called NETWORK#1 -> Print Status §

NETWORK#1 waiting to be proved.

Suggest proving VC called NETWORK#1 -> §

Entering Prover with verification condition NETWORK#1

```

H1: all I#1 : INTEGER [1..100],    AUSERPORT[I#1].SYSIN.INFROM = MSGSEQ()
      and AUSERPORT[I#1].SYSIN.OUTTO = MSGSEQ()
      and AUSERPORT[I#1].SYSOUT.INFROM = MSGSEQ()
      and AUSERPORT[I#1].SYSOUT.OUTTO = MSGSEQ()

H2: all I#2 : [1..NUSERS],
      all J : USERID,
      ( I#2 ne J
      -> AUSERPORT#1[J].SYSIN.PAR[I#2].INFROM = MSGSEQ())
      and AUSERPORT#1[J].SYSOUT.PAR[I#2].OUTTO
      sub SECUREMAIL(SEcurity[I#2], SEcurity[J],
      MAIL(AUSERPORT#1[I#2].SYSIN.PAR[I#2].INFROM,
      I#2, J))

H3: all I#3 : INTEGER [1..100],
      CO\AUSERPORT[I#3].SYSIN.INFROM @ AUSERPORT[I#3].SYSIN.INFROM
      = AUSERPORT#1[I#3].SYSIN.INFROM
      and CO\AUSERPORT[I#3].SYSIN.OUTTO @ AUSERPORT[I#3].SYSIN.OUTTO
      = AUSERPORT#1[I#3].SYSIN.OUTTO
      and CO\AUSERPORT[I#3].SYSOUT.INFROM @ AUSERPORT[I#3].SYSOUT.INFROM
      = AUSERPORT#1[I#3].SYSOUT.INFROM
      and CO\AUSERPORT[I#3].SYSOUT.OUTTO @ AUSERPORT[I#3].SYSOUT.OUTTO
      = AUSERPORT#1[I#3].SYSOUT.OUTTO

H4: all I#4 : INTEGER [1..100],
      ISMERGE(CO\AUSERPORT[I#4].SYSIN.INFROM,
      INFROM\ARRAYTOBAG(AUSERPORT#1[I#4].SYSIN.PAR, 1, NUSERS))
      and ISMERGE(CO\AUSERPORT[I#4].SYSIN.OUTTO,
      OUTTO\ARRAYTOBAG(AUSERPORT#1[I#4].SYSIN.PAR, 1, NUSERS))
      and ISMERGE(CO\AUSERPORT[I#4].SYSOUT.INFROM,
      INFROM\ARRAYTOBAG(AUSERPORT#1[I#4].SYSOUT.PAR, 1, NUSERS))
      and ISMERGE(CO\AUSERPORT[I#4].SYSOUT.OUTTO,
      OUTTO\ARRAYTOBAG(AUSERPORT#1[I#4].SYSOUT.PAR, 1, NUSERS))

-->
C1: all J : USERID, ISSUBMERGE(AUSERPORT#1[J].SYSOUT.OUTTO,
      ALLSECUREMAILFOR(AUSERPORT#1, J, SEcurity))

```

Only key parts of proofs are included. All properties used are explicitly noted; some are referred to in interactive dialogs.

Backup point
(.)

Occasionally, backup points and theorem labels, such as "(.)", are printed. Although they serve a useful purpose, they can be ignored here. The prover skolemizes Network#1 to eliminate quantifiers, yielding the following theorem.

Prover-> Print Theorem

```

H1. AUSERPORT[I#1$].SYSIN.INFROM = MSGSEQ()
H2. AUSERPORT[I#1$].SYSIN.OUTTO = MSGSEQ()
H3. AUSERPORT[I#1$].SYSOUT.INFROM = MSGSEQ()
H4. AUSERPORT[I#1$].SYSOUT.OUTTO = MSGSEQ()
H5. I#2$ ne J$ -> AUSERPORT#1[J$].SYSIN.PAR[I#2$].INFROM = MSGSEQ()
H6.  AUSERPORT#1[J$].SYSOUT.PAR[I#2$].OUTTO
    sub SECUREMAIL(SEcurity[I#2$], SEcurity[J$],
        MAIL(AUSERPORT#1[I#2$].SYSIN.PAR[I#2$].INFROM, I#2$, J$))
H7.  CO\AUSERPORT[I#3$].SYSIN.INFROM @ AUSERPORT[I#3$].SYSIN.INFROM
    = AUSERPORT#1[I#3$].SYSIN.INFROM
H8.  CO\AUSERPORT[I#3$].SYSIN.OUTTO @ AUSERPORT[I#3$].SYSIN.OUTTO
    = AUSERPORT#1[I#3$].SYSIN.OUTTO
H9.  CO\AUSERPORT[I#3$].SYSOUT.INFROM @ AUSERPORT[I#3$].SYSOUT.INFROM
    = AUSERPORT#1[I#3$].SYSOUT.INFROM
H10. CO\AUSERPORT[I#3$].SYSOUT.OUTTO @ AUSERPORT[I#3$].SYSOUT.OUTTO
    = AUSERPORT#1[I#3$].SYSOUT.OUTTO
H11. ISMERGE(CO\AUSERPORT[I#4$].SYSIN.INFROM,
    INFROM\ARRAYTOBAG(AUSERPORT#1[I#4$].SYSIN.PAR, 1, NUSERS))
H12. ISMERGE(CO\AUSERPORT[I#4$].SYSIN.OUTTO,
    OUTTO\ARRAYTOBAG(AUSERPORT#1[I#4$].SYSIN.PAR, 1, NUSERS))
H13. ISMERGE(CO\AUSERPORT[I#4$].SYSOUT.INFROM,
    INFROM\ARRAYTOBAG(AUSERPORT#1[I#4$].SYSOUT.PAR, 1, NUSERS))
H14. ISMERGE(CO\AUSERPORT[I#4$].SYSOUT.OUTTO,
    OUTTO\ARRAYTOBAG(AUSERPORT#1[I#4$].SYSOUT.PAR, 1, NUSERS))
IMP
C1. ISSUBMERGE(AUSERPORT#1[J].SYSOUT.OUTTO,
    ALLSECUREMAILFOR(AUSERPORT#1, J, SECURITY))

```

The user directs the prover to delete irrelevant hypotheses H2, H3, H8, H9, H12, and H13 and several equality substitutions are performed, resulting in the following theorem.

Prover-> Print Theorem

```

H1. I#2# ne J# -> AUSERPORT#1[J#].SYSIN.PAR[I#2#].INFROM = MSGSEQ()
H2. ISMERGE(AUSERPORT#1[I#1#].SYSIN.INFROM,
            INFROM\ARRAYTOBAG(AUSERPORT#1[I#1#].SYSIN.PAR, 1, NUSERS))
H3. ISMERGE(AUSERPORT#1[I#1#].SYSOUT.OUTTO,
            OUTTO\ARRAYTOBAG(AUSERPORT#1[I#1#].SYSOUT.PAR, 1, NUSERS))
H4. AUSERPORT#1[J#].SYSOUT.PAR[I#2#].OUTTO
    sub SECUREMAIL(SEcurity[I#2#], SECURITY[J#],
                  MAIL(AUSERPORT#1[I#2#].SYSIN.PAR[I#2#].INFROM, I#2#, J#))

```

IMP

```

C1. ISSUBMERGE(AUSERPORT#1[J].SYSOUT.OUTTO,
              ALLSECUREMAILFOR(AUSERPORT#1, J, SECURITY))

```

This cannot be proved without expanding the definition of IsSubmerge and adding a new lemma. These two steps are shown below, followed by a display of the transformed theorem.

Prover-> Expand IsSubmerge

Backup point

(. D PUT . =S . =S . E .)

Prover-> Print Conclusion

```

C1. ISMERGE(X#,
            ARRAYTOBAG(ALLSECUREMAILFOR(AUSERPORT#1, J, SECURITY), 1,
                      NUSERS))
C2. AUSERPORT#1[J].SYSOUT.OUTTO sub X#

```

Prover-> Use Lemma

Enter lemma ...

```

* all A: MsgBufArray, all B: MsgSeqArray, all X: MsgSeq,
*   IsMerge(X, Outto!\ArrayToBag(A, 1, Nusers))
*   and (all K: UserId, A[K].Outto sub B[K])
* -> some Y: MsgSeq,
*   IsMerge(Y, ArrayToBag(B, 1, Nusers))
*   and X sub Y;

```

Lemma added ... Its name is LEMMA#1

(. D PUT . =S . =S . E . U)

Prover-> Print Theorem

```

H1. A#[K].OUTTO sub B#[K]
    and ISMERGE(X#1, OUTTO\ARRAYTOBAG(A#, 1, NUSERS))
    -> ISMERGE(Y, ARRAYTOBAG(B#, 1, NUSERS)) and X#1 sub Y

```


Lemma#1 was skolemized and added as a new hypothesis, leaving the rest of the theorem unchanged. The prover backchains on H1, breaks the resulting theorem into two subgoals, then tries to prove the first subgoal and fails.

Prover-> Print Theorem

```

H1.  AUSERPORT#1[J$].SYSOUT.PAR[I#2$].OUTTO
      sub SECUREMAIL(SEcurity[I#2$], SECURITY[J$],
                     MAIL(AUSERPORT#1[I#2$].SYSIN.PAR[I#2$].INFROM, I#2$, J$))
H2. ISMERGE(AUSERPORT#1[I#1$].SYSOUT.OUTTO,
            OUTTO\ARRAYTOBAG(AUSERPORT#1[I#1$].SYSOUT.PAR, I, NUSERS))
H3. ISMERGE(AUSERPORT#1[I#1$].SYSIN.INFROM,
            INFROM\ARRAYTOBAG(AUSERPORT#1[I#1$].SYSIN.PAR, I, NUSERS))
H4. I#2$ ne J$ -> AUSERPORT#1[J$].SYSIN.PAR[I#2$].INFROM = MSGSEQ()
IMP
C1.  A$
      [K(AUSERPORT#1[J].SYSOUT.OUTTO,
        ALLSECUREMAILFOR(AUSERPORT#1, J, SECURITY), A$)]
      .OUTTO
      sub ALLSECUREMAILFOR(AUSERPORT#1, J, SECURITY)
        [K(AUSERPORT#1[J].SYSOUT.OUTTO,
          ALLSECUREMAILFOR(AUSERPORT#1, J, SECURITY), A$)]

```

The user directs the prover to establish an intermediate fact for use in the proof of C1.

Prover-> Claim

Newgoal:

* AUSERPORT#1[K].SYSIN.PAR[K].INFROM = AUSERPORT#1[K].SYSIN.INFROM;

After the user adds a lemma, the claim is established and then added as hypothesis H1.

Prover-> Use Lemma

Enter lemma . . .

```

* all A: MsgBufArray, all I: UserId, all X: MsgSeq,
*   (all J: UserId, I ne J -> A[J].Infrom=MsgSeq())
*   and IsMerge(X, Infrom!\ArrayToBag(A,I,Nusers))
* -> X = A[I].Infrom;

```

Lemma added . . . Its name is LEMMA#2

Prover-> Print Theorem

```
H1. AUSERPORT#1[K].SYSIN.PAR[K].INFROM = AUSERPORT#1[K].SYSIN.INFROM
H2.  AUSERPORT#1[J$].SYSOUT.PAR[#2$].OUTTO
    sub SECUREMAIL(SEcurity[#2$], SECURITY[J$],
        MAIL(AUSERPORT#1[#2$].SYSIN.PAR[#2$].INFROM, [#2$, J$])
H3. ISMERGE(AUSERPORT#1[#1$].SYSOUT.OUTTO,
    OUTTO\ARRAYTOBAG(AUSERPORT#1[#1$].SYSOUT.PAR, 1, NUSERS))
H4. ISMERGE(AUSERPORT#1[#1$].SYSIN.INFROM,
    INFROM\ARRAYTOBAG(AUSERPORT#1[#1$].SYSIN.PAR, 1, NUSERS))
H5. [#2$ ne J$ -> AUSERPORT#1[J$].SYSIN.PAR[#2$].INFROM = MSGSEQ()
IMP
C1.  A$
    [K(AUSERPORT#1[J].SYSOUT.OUTTO,
        ALLSECUREMAILFOR(AUSERPORT#1, J, SECURITY), A$)]
    .OUTTO
    sub ALLSECUREMAILFOR(AUSERPORT#1, J, SECURITY)
        [K(AUSERPORT#1[J].SYSOUT.OUTTO,
            ALLSECUREMAILFOR(AUSERPORT#1, J, SECURITY), A$)]
```

The remaining sequence of events is (a) the specifications of AllSecureMailFor are added, (b) the prover fails to prove the theorem, (c) the user directs the prover to build a chain of equalities, and (d) the prover tries again and succeeds.

NETWORK#1 proved in theorem prover.

Suggest fully defining SWITCHER-> Print Status All

The current design and verification status is:

Waiting for pending body to be filled in: SWITCHER

Proved: NETWORK

For specifications only: ARRAYTOBAG, ALLSECUREMAILFOR, ISSUBMERGE,
INFROM\ARRAYTOBAG, MAIL, OUTTO\ARRAYTOBAG, SECUREMAIL

Constants/Types: BAGMSGSEQ, MSG, MSGSEQ, MSGINBUF, MSGOUTBUF,
MSGBUFARRAY, MSGSEQARRAY, NUSERS, SECURITYARRAY, SECURITYCLASS,
USERID, USERPORT, USERPORTARRAY

As indicated by the status summary, the top-level network program is completely proved and the switcher process is waiting to be implemented. The user has SID preview the potential effects of several possible changes before proceeding.

Suggest fully defining SWITCHER-> Explain

<- What are the effects of filling in Switcher's pending body?

No effects.

<- Why?

Because the body of SWITCHER is currently undefined.

<- What are the effects of changing its block specification?

No effects on SWITCHER. The verification of other programs is not affected if the formula

Changed block specification

-->

```

C1: all J : USERID,      APORTS[J].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SECURITY[J],
                    MAIL(APORTS[I].SYSIN.INFROM, I, J))
      and (J ne I -> APORTS[J].SYSIN.INFROM = MSGSEQ(I))

```

is true. If not, the change invalidates the verification of NETWORK.

<- Why?

No effects on SWITCHER because it does not have any VCs. Invalid verification because the block specification of SWITCHER was used in proving NETWORK#1.

<- What are the effects of changing the header of SecureMail?

Possible effects of changing the header of SECUREMAIL are:

Incorrect semantics: ALLSECUREMAILFOR, SWITCHER

Invalid verifications: NETWORK

<- Why is Network affected?

Invalid verification because the external specifications of ALLSECUREMAILFOR, which call SECUREMAIL, were used in proving NETWORK#1. Invalid verification because the external specifications of SWITCHER, which call SECUREMAIL, were used in proving NETWORK#1.

<- What are the effects of changing its exit?

No effects on SECUREMAIL. The verification of other programs is not affected.

<- Why?

No effects on SECUREMAIL because it is for specifications only and does not have VCs. No external effects because properties from SECUREMAIL have not been used in proving any VCs.

<- What are the effects of changing Mail's exit assertion?

No effects on MAIL. The verification of other programs is not affected.

<- Why?

No effects on MAIL because it is for specifications only and does not have VCs. No external effects because properties from MAIL have not been used in proving any VCs.

<- Done

Suggest fully defining SWITCHER-> Edit

.

.

.

Exec-> Translate Swichr.Net

```

process Switcher(i:UserId; var aports:UserPortArray;
                  Security:SecurityArray) =
begin
  entry i in [1..NUsers];
  block all j:userid,
    aports(j).sysout.outto sub
      securemail(security(i),security(j),
        mail(aports(i).sysin.infrom,i,j))
    and
      ( j ne i -> aports(j).sysin.infrom = MsgSeq() );
  var ms : Msg := MakeMsg(i,i,MsgText());

  keep (assume source(ms) = i);
  loop
    assert all j:userid,
      aports(j).sysout.outto sub
        securemail(security(i),security(j),
          mail(aports(i).sysin.infrom,i,j))
      and
        ( j ne i -> aports(j).sysin.infrom = MsgSeq() );
    begin
      receive ms from aports(i).SysIn;
      if Secure(Security(i),Security(Destination(ms)))
      then send ms to aports(Destination(ms)).SysOut
      end;
    when
      is ReceiveError: ;
      is SendError: ;
    end;
  end;
end;

function SecureMail(s1,s2:SecurityClass;
                   ms:MsgSeq):MsgSeq =
begin
  exit (assume SecureMail(s1,s2,ms) =
    ( if Secure(s1,s2) then ms
      else MsgSeq() fi));
end;

function Mail(ms:MsgSeq;i,j:UserId):MsgSeq =
begin
  exit (assume Mail(ms,i,j)=

```

```

    ( if ms = MsgSeq() then MsgSeq()
      else if Source(last(ms)) = i and
            Destination(last(ms)) = j
            then Mail(nonlast(ms),i,j) @ MsgSeq(last(ms))
            else Mail(nonlast(ms),i,j) fi
      fi ));
end;

function Secure(s1,s2:SecurityClass):boolean = pending;

function Source(M:Msg):UserId = pending;
function Destination(M:Msg):UserId = pending;
function MakeMsg(S,D : UserId; T : MsgText) : Msg =
begin
  entry S in [1..NUsers] and D in [1..NUsers];
  pending
end;

type MsgText = pending;

```

No syntax errors detected.
 No semantic errors detected.

*The message switching process is completely implemented and specified.
 Several additional specifications are also entered for verification and type-
 checking purposes.*

Exec-> Print Status All

The current design and verification status is:

Waiting for VC generation: SWITCHER

Waiting for pending body to be filled in: DESTINATION, MAKEMSG, SECURE,
 SOURCE

Proved: NETWORK

For specifications only: ARRAYTOBAG, ALLSECUREMAILFOR, ISSUBMERGE,
 INFROM\ARRAYTOBAG, MAIL, OUTTO\ARRAYTOBAG, SECUREMAIL

Constants/Types: BAGMSGSEQ, MSG, MSGSEQ, MSGTEXT, MSGINBUF, MSGOUTBUF,
 MSGBUFARRAY, MSGSEQARRAY, NUSERS, SECURITYARRAY, SECURITYCLASS,
 USERID, USERPORT, USERPORTARRAY

Exec-> Suggest

Suggest generating VCs for SWITCHER-> 1

Generating VCs for PROCESS SWITCHER

Found 1-ST path

Found 2-ND path

Found 3-RD path

Note: Loop has no exit paths

Found 4-TH path

Found 5-TH path

Found 6-TH path

Found 7-TH path

The trace of how the VCs for Switcher are generated is covered in the next several pages.

Beginning new path...

APOINTS' := APOINTS

Assume Initial Buffer Histories are null

```

all I#1 : INTEGER [1..100],  APOINTS[I#1].SYSIN.INFROM = MSGSEQ()
                                and APOINTS[I#1].SYSIN.OUTTO = MSGSEQ()
                                and APOINTS[I#1].SYSOUT.INFROM = MSGSEQ()
                                and APOINTS[I#1].SYSOUT.OUTTO = MSGSEQ()

```

Assume (unit entry condition)

I in [1..NUSERS]

Initializing local variables

Evaluating MAKEMSG(I, I, MSGTEXT())

Continuing in path...

MS := MAKEMSG(I, I, MSGTEXT())

Assume (KEEP assertion)

SOURCE(MS) = I

Entering loop...

Evaluating SECUREMAIL(SEcurity[I], SEcurity[J#1],
MAIL(APOINTS[I].SYSIN.INFROM, I, J#1))

Continuing in path...

Evaluating MAIL(APORTS[I].SYSIN.INFROM, I, J#1)

Continuing in path...

```
ASSERT all J#1 : USERID,      APORTS[J#1].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
      MAIL(APORTS[I].SYSIN.INFROM, I,
      J#1))
      and (J#1 ne I -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())
```

Must verify ASSERT condition

Verification condition SWITCHER#1

```
H1: all I#1 : INTEGER [1..100],  APORTS[I#1].SYSIN.INFROM = MSGSEQ()
      and APORTS[I#1].SYSIN.OUTTO = MSGSEQ()
      and APORTS[I#1].SYSOUT.INFROM = MSGSEQ()
      and APORTS[I#1].SYSOUT.OUTTO = MSGSEQ()
```

H2: I = SOURCE(MAKEMSG(I, I, MSGTEXT()))

H3: I in [1..NUSERS]

-->

```
C1: all J#1 : USERID,  (I ne J#1 -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())
      and  APORTS[J#1].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
      MAIL(APORTS[I].SYSIN.INFROM, I, J#1))
```

End of path

Beginning new path...

Continuing in LOOP ...

Assume (from last assertion)

```
all J#1 : USERID,      APORTS[J#1].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
      MAIL(APORTS[I].SYSIN.INFROM, I, J#1))
      and (J#1 ne I -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())
```

Assume (KEEP assertion)

SOURCE(MS) = I

RECEIVE MS FROM APORTS[I].SYSIN
 MS := FIRST(APORTS#1[I].SYSIN.BUFQ)

Assume (KEEP assertion)
 SOURCE(MS) = I

Assume new buffer history

```
(all I#2 : INTEGER [1..100],
  APORTS[I#2].SYSIN.OTTO = APORTS#1[I#2].SYSIN.OTTO
  and APORTS[I#2].SYSOUT.INFROM = APORTS#1[I#2].SYSOUT.INFROM
  and APORTS[I#2].SYSOUT.OTTO = APORTS#1[I#2].SYSOUT.OTTO
  and ( I#2 ne I
    -> APORTS[I#2].SYSIN.INFROM = APORTS#1[I#2].SYSIN.INFROM))
and APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
  = APORTS#1[I].SYSIN.INFROM
```

APOINTS := APOINTS#1

Evaluating DESTINATION(MS)
 Continuing in path...

Evaluating SECURE(SEcurity[I], SECurity[DESTINATION(MS)])
 Continuing in path...

Assume (IF test succeeded)
 SECURE(SECURITY[I], SECURITY[DESTINATION(MS)])

SEND MS TO APOINTS[DESTINATION(MS)].SYSOUT

Assume new buffer history

```
all I#4 : INTEGER [1..100],    APOINTS#2[I#4].SYSIN.INFROM
                                = APOINTS[I#4].SYSIN.INFROM
                                and APOINTS#2[I#4].SYSIN.OTTO
                                = APOINTS[I#4].SYSIN.OTTO
                                and APOINTS#2[I#4].SYSOUT.INFROM
                                = APOINTS[I#4].SYSOUT.INFROM
                                and APOINTS#2[I#4].SYSOUT.OTTO
                                = APOINTS[I#4].SYSOUT.OTTO
```

APOINTS := APOINTS#2

Assume (SEND blocked)
 FULL(APOINTS[DESTINATION(MS)].SYSOUT)

Blockage assertion is

```

all J : USERID,      APORTS[J].SYSOUT.OUTTO
sub SECUREMAIL(SEcurity[I], SECURITY[J],
                MAIL(APORTS[I].SYSIN.INFROM, I, J))
and (J ne I -> APORTS[J].SYSIN.INFROM = MSGSEQ())

```

Must verify (process blockage) condition

Verification condition SWITCHER#2

```

H1: all I#2 : INTEGER [1..100],      APORTS[I#2].SYSIN.OUTTO
    = APORTS#1[I#2].SYSIN.OUTTO
    and APORTS[I#2].SYSOUT.INFROM
    = APORTS#1[I#2].SYSOUT.INFROM
    and APORTS[I#2].SYSOUT.OUTTO
    = APORTS#1[I#2].SYSOUT.OUTTO
    and ( I ne I#2
    -> APORTS[I#2].SYSIN.INFROM
    = APORTS#1[I#2].SYSIN.INFROM)

H2: all I#4 : INTEGER [1..100],      APORTS#1[I#4].SYSIN.INFROM
    = APORTS#2[I#4].SYSIN.INFROM
    and APORTS#1[I#4].SYSIN.OUTTO
    = APORTS#2[I#4].SYSIN.OUTTO
    and APORTS#1[I#4].SYSOUT.INFROM
    = APORTS#2[I#4].SYSOUT.INFROM
    and APORTS#1[I#4].SYSOUT.OUTTO
    = APORTS#2[I#4].SYSOUT.OUTTO

H3: all J#1 : USERID,      (I ne J#1 -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())
    and APORTS[J#1].SYSOUT.OUTTO
    sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
                    MAIL(APORTS[I].SYSIN.INFROM, I, J#1))

H4: APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
    = APORTS#1[I].SYSIN.INFROM
H5: I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))
H6: I = SOURCE(MS)
H7: FULL(APORTS#2[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))].SYSOUT)
H8: SECURE(SEcurity[I],
            SECURITY[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))])

```

-->

```

C1: all J : USERID,      (I ne J -> APORTS#2[J].SYSIN.INFROM = MSGSEQ())
    and APORTS#2[J].SYSOUT.OUTTO
    sub SECUREMAIL(SEcurity[I], SECURITY[J],
                    MAIL(APORTS#2[I].SYSIN.INFROM, I, J))

```

End of path

Beginning new path...
Continuing in LOOP ...

Assume (from last assertion)

```

all J#1 : USERID,      APORTS[J#1].SYSOUT.OUTTO
                        sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
                                MAIL(APORTS[I].SYSIN.INFROM, I, J#1))
and (J#1 ne I -> APORTS[J#1].SYSIN.INFROM = MSGSEQ(I))

```

Assume (KEEP assertion)

SOURCE(MS) = I

RECEIVE MS FROM APORTS[I].SYSIN

Assume new buffer history

```

all I#5 : INTEGER [1..100],      APORTS#1[I#5].SYSIN.INFROM
                                = APORTS[I#5].SYSIN.INFROM
and APORTS#1[I#5].SYSIN.OUTTO
                                = APORTS[I#5].SYSIN.OUTTO
and APORTS#1[I#5].SYSOUT.INFROM
                                = APORTS[I#5].SYSOUT.INFROM
and APORTS#1[I#5].SYSOUT.OUTTO
                                = APORTS[I#5].SYSOUT.OUTTO

```

APORTS := APORTS#1

Assume (RECEIVE blocked)

EMPTY(APORTS[I].SYSIN)

Blockage assertion is

```

all J : USERID,      APORTS[J].SYSOUT.OUTTO
                        sub SECUREMAIL(SEcurity[I], SECURITY[J],
                                MAIL(APORTS[I].SYSIN.INFROM, I, J))
and (J ne I -> APORTS[J].SYSIN.INFROM = MSGSEQ(I))

```

Must verify (process blockage) condition
Verification condition SWITCHER#3

```

H1: all I#5 : INTEGER [1..100],    APORTS[I#5].SYSIN.INFROM
    = APORTS#1[I#5].SYSIN.INFROM
    and APORTS[I#5].SYSIN.OUTTO
    = APORTS#1[I#5].SYSIN.OUTTO
    and APORTS[I#5].SYSOUT.INFROM
    = APORTS#1[I#5].SYSOUT.INFROM
    and APORTS[I#5].SYSOUT.OUTTO
    = APORTS#1[I#5].SYSOUT.OUTTO
H2: all J#1 : USERID,    (I ne J#1 -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())
    and APORTS[J#1].SYSOUT.OUTTO
    sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
        MAIL(APORTS[I].SYSIN.INFROM, I, J#1))
H3: EMPTY(APORTS#1[I].SYSIN)
H4: I = SOURCE(MS)
-->
C1: all J : USERID,    (I ne J -> APORTS#1[J].SYSIN.INFROM = MSGSEQ())
    and APORTS#1[J].SYSOUT.OUTTO
    sub SECUREMAIL(SEcurity[I], SECURITY[J],
        MAIL(APORTS#1[I].SYSIN.INFROM, I, J))

```

End of path

Beginning new path...
Continuing in LOOP ...

Assume (from last assertion)
 all J#1 : USERID, APORTS[J#1].SYSOUT.OUTTO
 sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
 MAIL(APORTS[I].SYSIN.INFROM, I, J#1))
 and (J#1 ne I -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())

Assume (KEEP assertion)
 SOURCE(MS) = I

RECEIVE MS FROM APORTS[I].SYSIN
 MS := FIRST(APORTS#1[I].SYSIN.BUFQ)

Assume (KEEP assertion)

SOURCE(MS) = I

Assume new buffer history

(all I#2 : INTEGER [1..100],

APORTS[I#2].SYSIN.OUTTO = APORTS#1[I#2].SYSIN.OUTTO

and APORTS[I#2].SYSOUT.INFROM = APORTS#1[I#2].SYSOUT.INFROM

and APORTS[I#2].SYSOUT.OUTTO = APORTS#1[I#2].SYSOUT.OUTTO

and (I#2 ne I

-> APORTS[I#2].SYSIN.INFROM = APORTS#1[I#2].SYSIN.INFROM))

and APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))

= APORTS#1[I].SYSIN.INFROM

APORTS := APORTS#1

Evaluating DESTINATION(MS)

Continuing in path...

Evaluating SECURE(SEcurity[I], SECURITY[DESTINATION(MS)])

Continuing in path...

Assume (IF test succeeded)

SECURE(SEcurity[I], SECURITY[DESTINATION(MS)])

SEND MS TO APORTS[DESTINATION(MS)].SYSOUT

Assume new buffer history

(all I#3 : INTEGER [1..100],

APORTS[I#3].SYSIN.INFROM = APORTS#2[I#3].SYSIN.INFROM

and APORTS[I#3].SYSIN.OUTTO = APORTS#2[I#3].SYSIN.OUTTO

and APORTS[I#3].SYSOUT.INFROM = APORTS#2[I#3].SYSOUT.INFROM

and (I#3 ne DESTINATION(MS)

-> APORTS[I#3].SYSOUT.OUTTO = APORTS#2[I#3].SYSOUT.OUTTO))

and APORTS[DESTINATION(MS)].SYSOUT.OUTTO @ MSGSEQ(MS)

= APORTS#2[DESTINATION(MS)].SYSOUT.OUTTO

APORTS := APORTS#2

Entering next iteration of loop...

Evaluating SECUREMAIL(SEcurity[I], SECURITY[J#1],

MAIL(APORTS[I].SYSIN.INFROM, I, J#1))

Continuing in path...

Evaluating MAIL(APORTS[I].SYSIN.INFROM, I, J#1)

Continuing in path...

```

ASSERT all J#1 : USERID,      APORTS[J#1].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
      MAIL(APORTS[I].SYSIN.INFROM, I,
      J#1))
      and (J#1 ne I -> APORTS[J#1].SYSIN.INFROM = MSGSEQ(I))

```

 Must verify ASSERT condition

Verification condition SWITCHER#4

```

H1: all I#2 : INTEGER [1..100],      APORTS[I#2].SYSIN.OUTTO
      = APORTS#1[I#2].SYSIN.OUTTO
      and APORTS[I#2].SYSOUT.INFROM
      = APORTS#1[I#2].SYSOUT.INFROM
      and APORTS[I#2].SYSOUT.OUTTO
      = APORTS#1[I#2].SYSOUT.OUTTO
      and ( I ne I#2
      -> APORTS[I#2].SYSIN.INFROM
      = APORTS#1[I#2].SYSIN.INFROM)

H2: all I#3 : INTEGER [1..100],
      APORTS#1[I#3].SYSIN.INFROM = APORTS#2[I#3].SYSIN.INFROM
      and APORTS#1[I#3].SYSIN.OUTTO = APORTS#2[I#3].SYSIN.OUTTO
      and APORTS#1[I#3].SYSOUT.INFROM = APORTS#2[I#3].SYSOUT.INFROM
      and ( I#3 ne DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))
      -> APORTS#1[I#3].SYSOUT.OUTTO = APORTS#2[I#3].SYSOUT.OUTTO)

H3: all J#1 : USERID,      (I ne J#1 -> APORTS[J#1].SYSIN.INFROM = MSGSEQ(I))
      and APORTS[J#1].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
      MAIL(APORTS[I].SYSIN.INFROM, I, J#1))

H4: APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
      = APORTS#1[I].SYSIN.INFROM

H5: APORTS#1[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))].SYSOUT.OUTTO
      @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
      = APORTS#2[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))].SYSOUT.OUTTO

H6: I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))

H7: I = SOURCE(MS)

H8: SECURE(SEcurity[I],
      SECURITY[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))])

```

-->

```

C1: all J#1 : USERID,      (I ne J#1 -> APORTS#2[J#1].SYSIN.INFROM = MSGSEQ(I))
      and APORTS#2[J#1].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
      MAIL(APORTS#2[I].SYSIN.INFROM, I,
      J#1))

```

 End of path

 Beginning new path...
 Continuing in LOOP ...

Assume (from last assertion)

```

all J#1 : USERID,      APORTS[J#1].SYSOUT.UTTO
                      sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
                      MAIL(APORTS[I].SYSIN.INFROM, I, J#1))
and (J#1 ne 1 -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())
  
```

Assume (KEEP assertion)

SOURCE(MS) = I

RECEIVE MS FROM APORTS[I].SYSIN
 MS := FIRST(APORTS#1[I].SYSIN.BUFQ)

Assume (KEEP assertion)

SOURCE(MS) = I

Assume new buffer history

```

(all I#2 : INTEGER [1..100],
  APORTS[I#2].SYSIN.UTTO = APORTS#1[I#2].SYSIN.UTTO
  and APORTS[I#2].SYSOUT.INFROM = APORTS#1[I#2].SYSOUT.INFROM
  and APORTS[I#2].SYSOUT.UTTO = APORTS#1[I#2].SYSOUT.UTTO
  and ( I#2 ne 1
    -> APORTS[I#2].SYSIN.INFROM = APORTS#1[I#2].SYSIN.INFROM))
and APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
= APORTS#1[I].SYSIN.INFROM
  
```

APORTS := APORTS#1

Evaluating DESTINATION(MS)

Continuing in path...

Evaluating SECURE(SECURITY[I], SECURITY[DESTINATION(MS)])

Continuing in path...

Assume (IF test succeeded)

SECURE(SEcurity[I], SECURITY[DESTINATION(MS)])

Evaluating DESTINATION(MS)

Continuing in path...

SEND MS TO APORTS[DESTINATION(MS)].SYSOUT

Signalling condition SENDERROR

Entering WHEN part...

Entering next iteration of loop...

Evaluating SECUREMAIL(SEcurity[I], SECURITY[J#1],

MAIL(APORTS[I].SYSIN.INFROM, I, J#1))

Continuing in path...

Evaluating MAIL(APORTS[I].SYSIN.INFROM, I, J#1)

Continuing in path...

ASSERT all J#1 : USERID, APORTS[J#1].SYSOUT.OUTTO

sub SECUREMAIL(SEcurity[I], SECURITY[J#1],

MAIL(APORTS[I].SYSIN.INFROM, I,

J#1))

and (J#1 ne I -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())

Must verify ASSERT condition

Verification condition SWITCHER#5

H1: all I#2 : INTEGER [1..100], APORTS[I#2].SYSIN.OUTTO

= APORTS#1[I#2].SYSIN.OUTTO

and APORTS[I#2].SYSOUT.INFROM

= APORTS#1[I#2].SYSOUT.INFROM

and APORTS[I#2].SYSOUT.OUTTO

= APORTS#1[I#2].SYSOUT.OUTTO

and (I ne I#2

-> APORTS[I#2].SYSIN.INFROM

= APORTS#1[I#2].SYSIN.INFROM)

H2: all J#1 : USERID, (I ne J#1 -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())

and APORTS[J#1].SYSOUT.OUTTO

sub SECUREMAIL(SEcurity[I], SECURITY[J#1],

MAIL(APORTS[I].SYSIN.INFROM, I, J#1))

H3: APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))

= APORTS#1[I].SYSIN.INFROM

H4: I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))

H5: I = SOURCE(MS)

H6: SECURE(SEcurity[I],

SECURITY[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))])

-->

C1: all J#1 : USERID, (I ne J#1 -> APORTS#1[J#1].SYSIN.INFROM = MSGSEQ())
 and APORTS#1[J#1].SYSOUT.OUTTO
 sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
 MAIL(APORTS#1[I].SYSIN.INFROM, I,
 J#1))

 End of path

 Beginning new path...
 Continuing in LOOP ...

Assume (from last assertion)

all J#1 : USERID, APORTS[J#1].SYSOUT.OUTTO
 sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
 MAIL(APORTS[I].SYSIN.INFROM, I, J#1))
 and (J#1 ne I -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())

Assume (KEEP assertion)

SOURCE(MS) = I

RECEIVE MS FROM APORTS[I].SYSIN
 MS := FIRST(APORTS#1[I].SYSIN.BUFQ)

Assume (KEEP assertion)

SOURCE(MS) = I

Assume new buffer history

(all I#2 : INTEGER [1..100],
 APORTS[I#2].SYSIN.OUTTO = APORTS#1[I#2].SYSIN.OUTTO
 and APORTS[I#2].SYSOUT.INFROM = APORTS#1[I#2].SYSOUT.INFROM
 and APORTS[I#2].SYSOUT.OUTTO = APORTS#1[I#2].SYSOUT.OUTTO
 and (I#2 ne I
 -> APORTS[I#2].SYSIN.INFROM = APORTS#1[I#2].SYSIN.INFROM))
 and APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
 = APORTS#1[I].SYSIN.INFROM

APORTS := APORTS#1

Assume (IF test failed)

(not SECURE(SEcurity[I], SECURITY[DESTINATION(MS))])

Entering next iteration of loop...

**Evaluating SECUREMAIL(SECURITY[I], SECURITY[J#1],
MAIL(APORTS[I].SYSIN.INFROM, I, J#1))**

Continuing in path...

Evaluating MAIL(APORTS[I].SYSIN.INFROM, I, J#1)

Continuing in path...

**ASSERT all J#1 : USERID, APORTS[J#1].SYSOUT.OUTTO
sub SECUREMAIL(SECURITY[I], SECURITY[J#1],
MAIL(APORTS[I].SYSIN.INFROM, I,
J#1))
and (J#1 ne I -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())**

Must verify ASSERT condition

Verification condition SWITCHER#6

**H1: all I#2 : INTEGER [1..100], APORTS[I#2].SYSIN.OUTTO
= APORTS#1[I#2].SYSIN.OUTTO
and APORTS[I#2].SYSOUT.INFROM
= APORTS#1[I#2].SYSOUT.INFROM
and APORTS[I#2].SYSOUT.OUTTO
= APORTS#1[I#2].SYSOUT.OUTTO
and (I ne I#2
-> APORTS[I#2].SYSIN.INFROM
= APORTS#1[I#2].SYSIN.INFROM)**

**H2: all J#1 : USERID, (I ne J#1 -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())
and APORTS[J#1].SYSOUT.OUTTO
sub SECUREMAIL(SECURITY[I], SECURITY[J#1],
MAIL(APORTS[I].SYSIN.INFROM, I, J#1))**

**H3: APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
= APORTS#1[I].SYSIN.INFROM**

H4: I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))

H5: I = SOURCE(MS)

**H6: (not SECURE(SECURITY[I],
SECURITY[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))]))**

-->

**C1: all J#1 : USERID, (I ne J#1 -> APORTS#1[J#1].SYSIN.INFROM = MSGSEQ())
and APORTS#1[J#1].SYSOUT.OUTTO**

```

sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
               MAIL(APORTS#1[I].SYSIN.INFROM, I,
                   J#1))

```

End of path

Beginning new path...
Continuing in LOOP ...

Assume (from last assertion)

```

all J#1 : USERID,      APORTS[J#1].SYSOUT.OUTTO
sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
               MAIL(APORTS[I].SYSIN.INFROM, I, J#1))
and (J#1 ne 1 -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())

```

Assume (KEEP assertion)
SOURCE(MS) = 1

RECEIVE MS FROM APORTS[I].SYSIN

Assume new buffer history

```

(all I#2 : INTEGER [1..100],
  APORTS[I#2].SYSIN.OUTTO = APORTS#1[I#2].SYSIN.OUTTO
  and APORTS[I#2].SYSOUT.INFROM = APORTS#1[I#2].SYSOUT.INFROM
  and APORTS[I#2].SYSOUT.OUTTO = APORTS#1[I#2].SYSOUT.OUTTO
  and ( I#2 ne 1
    -> APORTS[I#2].SYSIN.INFROM = APORTS#1[I#2].SYSIN.INFROM))
and APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
= APORTS#1[I].SYSIN.INFROM

```

Signalling condition RECEIVEERROR

Entering WHEN part...

Entering next iteration of loop...

Evaluating SECUREMAIL(SEcurity[I], SECURITY[J#1],
MAIL(APORTS[I].SYSIN.INFROM, I, J#1))
Continuing in path...

Evaluating MAIL(APOINTS[I].SYSIN.INFROM, I, J#1)

Continuing in path...

```
ASSERT all J#1 : USERID,      APOINTS[J#1].SYSOUT.OUTTO
                        sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
                        MAIL(APOINTS[I].SYSIN.INFROM, I,
                        J#1))
                        and (J#1 ne I -> APOINTS[J#1].SYSIN.INFROM = MSGSEQ(I))
```

Must verify ASSERT condition

VC is trivially TRUE

End of path

Suggest proving VC called SWITCHER#1 -> Print Status All

The current design and verification status is:

Waiting for pending body to be filled in: DESTINATION, MAKEMSG, SECURE,
SOURCE

Proved: NETWORK

For specifications only: ARRAYTOBAG, ALLSECUREMAILFOR, ISSUBMERGE,
INFROM\ARRAYTOBAG, MAIL, OUTTO\ARRAYTOBAG, SECUREMAIL

Constants/Types: BAGMSGSEQ, MSG, MSGSEQ, MSGTEXT, MSGINBUF, MSGOUTBUF,
MSGBUFARRAY, MSGSEQARRAY, NUSERS, SECURITYARRAY, SECURITYCLASS,
USERID, USERPORT, USERPORTARRAY

SWITCHER

Waiting to be proved: SWITCHER#1, SWITCHER#2, SWITCHER#3, SWITCHER#4,
SWITCHER#5, SWITCHER#6

Proved in VC generator: SWITCHER#7

*As reflected in the status summary, seven VCs were generated for Switcher, six
of which are unproved. They are proved in sequence below.*

Suggest proving VC called SWITCHER#1 -> §

Entering Prover with verification condition SWITCHER#1

```
H1: all I#1 : INTEGER [1..100],  APOINTS[I#1].SYSIN.INFROM = MSGSEQ(I)
                        and APOINTS[I#1].SYSIN.OUTTO = MSGSEQ(I)
                        and APOINTS[I#1].SYSOUT.INFROM = MSGSEQ(I)
                        and APOINTS[I#1].SYSOUT.OUTTO = MSGSEQ(I)
```

H2: I = SOURCE(MAKEMSG(I, I, MSGTEXT()))

H3: I in [1..NUSERS]

-->

```

C1: all J#1 : USERID,    (I ne J#1 -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())
    and  APORTS[J#1].SYSOUT.OUTTO
    sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
        MAIL(APORTS[I].SYSIN.INFROM, I, J#1))

```

SWITCHER#1 proved in theorem prover.

This proof was done automatically except for a few interactive equality substitutions.

Suggest proving VC called SWITCHER#2-> §

Entering Prover with verification condition SWITCHER#2

```

H1: all I#2 : INTEGER [1..100],    APORTS[I#2].SYSIN.OUTTO
    = APORTS#1[I#2].SYSIN.OUTTO
    and  APORTS[I#2].SYSOUT.INFROM
    = APORTS#1[I#2].SYSOUT.INFROM
    and  APORTS[I#2].SYSOUT.OUTTO
    = APORTS#1[I#2].SYSOUT.OUTTO
    and ( I ne I#2
    ->  APORTS[I#2].SYSIN.INFROM
    = APORTS#1[I#2].SYSIN.INFROM)

H2: all I#4 : INTEGER [1..100],    APORTS#1[I#4].SYSIN.INFROM
    = APORTS#2[I#4].SYSIN.INFROM
    and  APORTS#1[I#4].SYSIN.OUTTO
    = APORTS#2[I#4].SYSIN.OUTTO
    and  APORTS#1[I#4].SYSOUT.INFROM
    = APORTS#2[I#4].SYSOUT.INFROM
    and  APORTS#1[I#4].SYSOUT.OUTTO
    = APORTS#2[I#4].SYSOUT.OUTTO

H3: all J#1 : USERID,    (I ne J#1 -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())
    and  APORTS[J#1].SYSOUT.OUTTO
    sub SECUREMAIL(SEcurity[I], SECURITY[J#1],
        MAIL(APORTS[I].SYSIN.INFROM, I, J#1))

H4:  APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
    = APORTS#1[I].SYSIN.INFROM
H5: I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))
H6: I = SOURCE(MS)

```

H7: FULL(APORTS#2[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))].SYSOUT)

H8: SECURE(SEcurity[I],
SECURITY[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))])

-->

C1: all J : USERID, (I ne J -> APORTS#2[J].SYSIN.INFROM = MSGSEQ())
and APORTS#2[J].SYSOUT.OUTTO
sub SECUREMAIL(SECURITY[I], SECURITY[J],
MAIL(APORTS#2[I].SYSIN.INFROM, I, J))

The proof of this theorem is representative of many of the other proofs and is therefore presented in its entirety. The current theorem is printed frequently to aid in following the manipulations.

Backup point

(.)

Prover-> Print Theorem

H1. I

in [MAX(SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ)), SOURCE(MS))
..MIN(SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ)), SOURCE(MS))]

H2. APORTS[I#2\$].SYSIN.OUTTO = APORTS#1[I#2\$].SYSIN.OUTTO

H3. APORTS[I#2\$].SYSOUT.INFROM = APORTS#1[I#2\$].SYSOUT.INFROM

H4. APORTS[I#2\$].SYSOUT.OUTTO = APORTS#1[I#2\$].SYSOUT.OUTTO

H5. I ne I#2\$ -> APORTS[I#2\$].SYSIN.INFROM = APORTS#1[I#2\$].SYSIN.INFROM

H6. APORTS#1[I#4\$].SYSIN.INFROM = APORTS#2[I#4\$].SYSIN.INFROM

H7. APORTS#1[I#4\$].SYSIN.OUTTO = APORTS#2[I#4\$].SYSIN.OUTTO

H8. APORTS#1[I#4\$].SYSOUT.INFROM = APORTS#2[I#4\$].SYSOUT.INFROM

H9. APORTS#1[I#4\$].SYSOUT.OUTTO = APORTS#2[I#4\$].SYSOUT.OUTTO

H10. I ne J#1\$ -> APORTS[J#1\$].SYSIN.INFROM = MSGSEQ()

H11. APORTS[J#1\$].SYSOUT.OUTTO

sub SECUREMAIL(SECURITY[I], SECURITY[J#1\$],
MAIL(APORTS[I].SYSIN.INFROM, I, J#1\$))

H12. APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
= APORTS#1[I].SYSIN.INFROM

H13. I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))

H14. I = SOURCE(MS)

H15. FULL(APORTS#2[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))].SYSOUT)

H16. SECURE(SECURITY[I],
SECURITY[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))])

IMP

C1. I ne J -> APORTS#2[J].SYSIN.INFROM = MSGSEQ()

C2. APORTS#2[J].SYSOUT.OUTTO

```
sub SECUREMAIL(SEcurity[I], SECURITY[J],
  MAIL(APORTS#2[I].SYSIN.INFROM, I, J))
```

Prover-> Delete H2,H3,H7,H8

Several irrelevant hypotheses are deleted.

Prover-> Proceeding

(. D I)

Backup point

(. D I P-> .)

Prover-> Print Theorem

H1. I ne J

H2. I

```
in [MAX(SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ)), SOURCE(MS))
  ..MIN(SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ)), SOURCE(MS))]
```

H3. APORTS[I#2\$].SYSOUT.OUTTO = APORTS#1[I#2\$].SYSOUT.OUTTO

H4. I ne I#2\$ -> APORTS[I#2\$].SYSIN.INFROM = APORTS#1[I#2\$].SYSIN.INFROM

H5. APORTS#1[I#4\$].SYSIN.INFROM = APORTS#2[I#4\$].SYSIN.INFROM

H6. APORTS#1[I#4\$].SYSOUT.OUTTO = APORTS#2[I#4\$].SYSOUT.OUTTO

H7. I ne J#1\$ -> APORTS[J#1\$].SYSIN.INFROM = MSGSEQ()

H8. APORTS[J#1\$].SYSOUT.OUTTO

```
sub SECUREMAIL(SEcurity[I], SECURITY[J#1$],
  MAIL(APORTS[I].SYSIN.INFROM, I, J#1$))
```

H9. APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
= APORTS#1[I].SYSIN.INFROM

H10. I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))

H11. I = SOURCE(MS)

H12. FULL(APORTS#2[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))].SYSOUT)

H13. SECURE(SEcurity[I],
SECURITY[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))])

IMP

C1. APORTS#2[J].SYSIN.INFROM = MSGSEQ()

The prover split the theorem into two subgoals and promoted the hypothesis of C1 to hypothesis H1 of the theorem. Hypotheses not relevant to this subgoal are deleted, retaining only four.

Prover-> Retain H1,H4,H5,H7

Prover-> Print Theorem

H1. I ne J
H2. I ne I#2\$ -> APORTS[I#2\$].SYSIN.INFROM = APORTS#1[I#2\$].SYSIN.INFROM
H3. APORTS#1[I#4\$].SYSIN.INFROM = APORTS#2[I#4\$].SYSIN.INFROM
H4. I ne J#1\$ -> APORTS[J#1\$].SYSIN.INFROM = MSGSEQ()

IMP

C1. APORTS#2[J].SYSIN.INFROM = MSGSEQ()

The proof of this theorem requires establishing a chain of equalities involving H3, C1, and the conclusions of H2 and H4.

Prover-> Eq Chain

New subgoal

Backup point

(. D 1 P-> . D =CHAIN .)

Prover-> Print Conclusion

C1. I ne J

C2. I ne J

C1 and C2 are hypotheses from H2 and H4 that must be proved for the chain to be valid.

Prover-> Simplify Theorem

Prover-> Print Conclusion

C1. TRUE

This completes the proof of subgoal C1. The prover tries and fails to prove C2, which cannot be proved without an additional lemma.

Prover-> \$Proceeding

(. D 2)

.....Ran out of tricks

Prover-> Print Theorem

H1. I

in [MAX(SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ)), SOURCE(MS))
..MIN(SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ)), SOURCE(MS))]

H2. APORTS[I#2\$].SYSOUT.OUTTO = APORTS#1[I#2\$].SYSOUT.OUTTO

H3. I ne I#2\$ -> APORTS[I#2\$].SYSIN.INFROM = APORTS#1[I#2\$].SYSIN.INFROM


```

H4. APORTS#1[I#4$].SYSIN.INFROM = APORTS#2[I#4$].SYSIN.INFROM
H5. APORTS#1[I#4$].SYSOUT.OUTTO = APORTS#2[I#4$].SYSOUT.OUTTO
H6. I ne J#1$ -> APORTS[J#1$].SYSIN.INFROM = MSGSEQ()
H7. APORTS[J#1$].SYSOUT.OUTTO
    sub SECUREMAIL(SEcurity[I], SEcurity[J#1$],
        MAIL(APORTS[I].SYSIN.INFROM, I, J#1$))
H8. APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
    = APORTS#1[I].SYSIN.INFROM
H9. I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))
H10. I = SOURCE(MS)
H11. FULL(APORTS#2[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))].SYSOUT)
H12. SECURE(SEcurity[I],
    SEcurity[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))])
IMP
C1. APORTS#2[J].SYSOUT.OUTTO
    sub SECUREMAIL(SEcurity[I], SEcurity[J],
        MAIL(APORTS#2[I].SYSIN.INFROM, I, J))

```

Prover-> Use Lemma

Enter lemma . . .

* all A,B: MsgSeq, all C:Msg, all K,L: UserId, all S1,S2: SecurityClass,

* A sub SecureMail(S1,S2,Mail(B,K,L))

* -> A sub SecureMail(S1,S2,Mail(B@MsgSeq(C),K,L));

Lemma added . . . Its name is LEMMA#3

(. D 2 U)

Prover-> Print Theorem

```

H1. A$ sub SECUREMAIL(S1$, S2$, MAIL(B$, K$, L$))
    -> A$ sub SECUREMAIL(S1$, S2$, MAIL(B$@MSGSEQ(C$), K$, L$))
H2. I
    in [MAX(SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ)), SOURCE(MS))
        ..MIN(SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ)), SOURCE(MS))]
H3. APORTS[I#2$].SYSOUT.OUTTO = APORTS#1[I#2$].SYSOUT.OUTTO
H4. I ne I#2$ -> APORTS[I#2$].SYSIN.INFROM = APORTS#1[I#2$].SYSIN.INFROM
H5. APORTS#1[I#4$].SYSIN.INFROM = APORTS#2[I#4$].SYSIN.INFROM
H6. APORTS#1[I#4$].SYSOUT.OUTTO = APORTS#2[I#4$].SYSOUT.OUTTO
H7. I ne J#1$ -> APORTS[J#1$].SYSIN.INFROM = MSGSEQ()
H8. APORTS[J#1$].SYSOUT.OUTTO
    sub SECUREMAIL(SEcurity[I], SEcurity[J#1$],
        MAIL(APORTS[I].SYSIN.INFROM, I, J#1$))
H9. APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
    = APORTS#1[I].SYSIN.INFROM

```

H10. I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))
H11. I = SOURCE(MS)
H12. FULL(APORTS#2[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))].SYSOUT)
H13. SECURE(SEcurity[I],
SECURITY[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))])

IMP

C1. APORTS#2[J].SYSOUT.OUTTO
sub SECUREMAIL(SEcurity[I], SECURITY[J],
MAIL(APORTS#2[I].SYSIN.INFROM, I, J))

Lemma#3 has been added to the theorem as hypothesis H1. Two equality chains are needed to complete the proof.

Prover-> Eq Chain

New subgoal

Backup point

(. D 2 U . =CHAIN .)

Prover-> Print Conclusion

C1. APORTS#2[J].SYSOUT.OUTTO
sub SECUREMAIL(SEcurity[I], SECURITY[J],
MAIL(APORTS[I].SYSIN.INFROM, I, J))

Prover-> Retain H3,H6,H8

Prover-> Print Theorem

H1. APORTS[I#2\$].SYSOUT.OUTTO = APORTS#1[I#2\$].SYSOUT.OUTTO
H2. APORTS#1[I#4\$].SYSOUT.OUTTO = APORTS#2[I#4\$].SYSOUT.OUTTO
H3. APORTS[J#1\$].SYSOUT.OUTTO
sub SECUREMAIL(SEcurity[I], SECURITY[J#1\$],
MAIL(APORTS[I].SYSIN.INFROM, I, J#1\$))

IMP

C1. APORTS#2[J].SYSOUT.OUTTO
sub SECUREMAIL(SEcurity[I], SECURITY[J],
MAIL(APORTS[I].SYSIN.INFROM, I, J))

Prover-> Eq Chain

New subgoal

Backup point

(. D 2 U . =CHAIN . D =CHAIN .)

Prover-> Print Conclusion

C1. TRUE

Prover-> Proceeding

SWITCHER#2 proved in theorem prover.

Suggest proving VC called SWITCHER#3-> 1

Entering Prover with verification condition SWITCHER#3

```

H1: all I#5 : INTEGER [1..100],    APORTS[I#5].SYSIN.INFROM
      = APORTS#1[I#5].SYSIN.INFROM
      and APORTS[I#5].SYSIN.OUTTO
      = APORTS#1[I#5].SYSIN.OUTTO
      and APORTS[I#5].SYSOUT.INFROM
      = APORTS#1[I#5].SYSOUT.INFROM
      and APORTS[I#5].SYSOUT.OUTTO
      = APORTS#1[I#5].SYSOUT.OUTTO
H2: all J#1 : USERID,    (I ne J#1 -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())
      and APORTS[J#1].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SEcurity[J#1],
        MAIL(APORTS[I].SYSIN.INFROM, I, J#1))
H3: EMPTY(APORTS#1[I].SYSIN)
H4: I = SOURCE(MS)
-->
C1: all J : USERID,    (I ne J -> APORTS#1[J].SYSIN.INFROM = MSGSEQ())
      and APORTS#1[J].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SEcurity[J],
        MAIL(APORTS#1[I].SYSIN.INFROM, I, J))

```

The proof of this VC is entirely automatic, with the user initiating two equality chains.

SWITCHER#3 proved in theorem prover.

Suggest proving VC called SWITCHER#4-> 1

Entering Prover with verification condition SWITCHER#4

```

H1: all I#2 : INTEGER [1..100],    APORTS[I#2].SYSIN.OUTTO
      = APORTS#1[I#2].SYSIN.OUTTO

```

```

and APORTS[I#2].SYSOUT.INFROM
  = APORTS#1[I#2].SYSOUT.INFROM
and APORTS[I#2].SYSOUT.OUTTO
  = APORTS#1[I#2].SYSOUT.OUTTO
and ( I ne I#2
  -> APORTS[I#2].SYSIN.INFROM
    = APORTS#1[I#2].SYSIN.INFROM)

H2: all I#3 : INTEGER [1..100],
    APORTS#1[I#3].SYSIN.INFROM = APORTS#2[I#3].SYSIN.INFROM
    and APORTS#1[I#3].SYSIN.OUTTO = APORTS#2[I#3].SYSIN.OUTTO
    and APORTS#1[I#3].SYSOUT.INFROM = APORTS#2[I#3].SYSOUT.INFROM
    and ( I#3 ne DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))
      -> APORTS#1[I#3].SYSOUT.OUTTO = APORTS#2[I#3].SYSOUT.OUTTO)

H3: all J#1 : USERID, (I ne J#1 -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())
    and APORTS[J#1].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SEcurity[J#1],
        MAIL(APORTS[I].SYSIN.INFROM, I, J#1))

H4: APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
    = APORTS#1[I].SYSIN.INFROM

H5: APORTS#1[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))].SYSOUT.OUTTO
    @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
    = APORTS#2[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))].SYSOUT.OUTTO

H6: I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))
H7: I = SOURCE(MS)
H8: SECURE(SEcurity[I],
    SEcurity[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))])

-->

C1: all J#1 : USERID, (I ne J#1 -> APORTS#2[J#1].SYSIN.INFROM = MSGSEQ())
    and APORTS#2[J#1].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SEcurity[J#1],
        MAIL(APORTS#2[I].SYSIN.INFROM, I,
          J#1))

```

The first subgoal was proved with a promotion and an equality chain. The proof of the second requires a cases argument, as shown below.

Prover-> Print Theorem


```

H1.  I
    in [MAX(SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ)), SOURCE(MS))
      ..MIN(SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ)), SOURCE(MS))]
H2. APORTS[I#2#1].SYSIN.OUTTO = APORTS#1[I#2#1].SYSIN.OUTTO
H3. APORTS[I#2#1].SYSOUT.INFROM = APORTS#1[I#2#1].SYSOUT.INFROM
H4. APORTS[I#2#1].SYSOUT.OUTTO = APORTS#1[I#2#1].SYSOUT.OUTTO
H5.  I ne I#2#1
    -> APORTS[I#2#1].SYSIN.INFROM = APORTS#1[I#2#1].SYSIN.INFROM
H6. APORTS#1[I#3#1].SYSIN.INFROM = APORTS#2[I#3#1].SYSIN.INFROM
H7. APORTS#1[I#3#1].SYSIN.OUTTO = APORTS#2[I#3#1].SYSIN.OUTTO
H8. APORTS#1[I#3#1].SYSOUT.INFROM = APORTS#2[I#3#1].SYSOUT.INFROM
H9.  DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ)) ne I#3#1
    -> APORTS#1[I#3#1].SYSOUT.OUTTO = APORTS#2[I#3#1].SYSOUT.OUTTO
H10. I ne J#1#2 -> APORTS[J#1#2].SYSIN.INFROM = MSGSEQ()
H11.  APORTS[J#1#2].SYSOUT.OUTTO
    sub SECUREMAIL(SEcurity[I], SEcurity[J#1#2],
      MAIL(APORTS[I].SYSIN.INFROM, I, J#1#2))
H12.  APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
    = APORTS#1[I].SYSIN.INFROM
H13.  APORTS#1[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))].SYSOUT
      .OUTTO
    @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
    = APORTS#2[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))].SYSOUT.OUTTO
H14. I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))
H15. I = SOURCE(MS)
H16. SECURE(SEcurity[I],
      SEcurity[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))])
IMP
C1.  APORTS#2[J#1].SYSOUT.OUTTO
    sub SECUREMAIL(SEcurity[I], SEcurity[J#1],
      MAIL(APORTS#2[I].SYSIN.INFROM, I, J#1))

```

Prover-> Cases

CASE:

* J#1 = DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ));

CASE:

* § DONE

Attempting CASE1

The remaining "not equal" case is generated by the prover. After deleting some irrelevant hypotheses and doing several equality substitutions, we come to the key part of the proof.

Prover-> Print Theorem

```

H1. APORTS#1[J#1].SYSOUT.OUTTO @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
    = APORTS#2[J#1].SYSOUT.OUTTO
H2. APORTS[J#1].SYSOUT.OUTTO = APORTS#1[J#1].SYSOUT.OUTTO
H3. J#1 = DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))
H4. I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))
H5. I = SOURCE(MS)
H6. SECURE(SECURITY[I], SECURITY[J#1])
H7. I ne J#1 -> APORTS#1[I].SYSOUT.OUTTO = APORTS#2[I].SYSOUT.OUTTO
H8. APORTS[J#1].SYSOUT.OUTTO
    sub SECUREMAIL(SECURITY[I], SECURITY[J#1],
        MAIL(APORTS[I].SYSIN.INFROM, I, J#1))
IMP
C1. APORTS#2[J#1].SYSOUT.OUTTO
    sub SECUREMAIL(SECURITY[I], SECURITY[J#1],
        MAIL( APORTS[I].SYSIN.INFROM
            @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ)), I, J#1))

```

The proof of this theorem requires the definitions of SecureMail and Mail.

Prover-> Expand SecureMail

More than one to expand.

Which do you want to expand? All

Backup point

(. 2 . CASE1 . D PUT . =S . E .)

Prover-> Print Theorem

```

H1. I
    in [MAX(SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ)), SOURCE(MS))
        ..MIN(SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ)), SOURCE(MS))]
H2. APORTS[J#1].SYSOUT.OUTTO
    sub if SECURE(SECURITY[I], SECURITY[J#1])
        then MAIL(APORTS[I].SYSIN.INFROM, I, J#1) else MSGSEQ() fi
H3. I ne J#1 -> APORTS#1[I].SYSOUT.OUTTO = APORTS#2[I].SYSOUT.OUTTO
H4. SECURE(SECURITY[I], SECURITY[J#1])
H5. I = SOURCE(MS)
H6. I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))
H7. J#1 = DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))
H8. APORTS[J#1].SYSOUT.OUTTO = APORTS#1[J#1].SYSOUT.OUTTO
H9. APORTS#1[J#1].SYSOUT.OUTTO @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
    = APORTS#2[J#1].SYSOUT.OUTTO

```

IMP

```

C1.  APORTS#2[J#1].SYSOUT.OUTTO
      sub if SECURE(SEcurity[I], SECURITY[J#1])
        then MAIL( APORTS[I].SYSIN.INFROM
                   @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ)), I, J#1)
        else MSGSEQ() fi

```

Prover-> Expand Mail

More than one to expand.

Which do you want to expand? Show

1. MAIL(APORTS[I].SYSIN.INFROM, I, J#1)
2. MAIL(APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ)), I, J#1)

Which do you want to expand? 2

Backup point

(2 . CASE1 . D PUT . =S . E . E .)

Prover-> Print Theorem

```

H1.  APORTS#1[J#1].SYSOUT.OUTTO @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
      = APORTS#2[J#1].SYSOUT.OUTTO
H2.  APORTS[J#1].SYSOUT.OUTTO = APORTS#1[J#1].SYSOUT.OUTTO
H3.  J#1 = DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))
H4.  I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))
H5.  I = SOURCE(MS)
H6.  SECURE(SEcurity[I], SECURITY[J#1])
H7.  I ne J#1 -> APORTS#1[I].SYSOUT.OUTTO = APORTS#2[I].SYSOUT.OUTTO
H8.  APORTS[J#1].SYSOUT.OUTTO
      sub if SECURE(SEcurity[I], SECURITY[J#1])
        then MAIL(APORTS[I].SYSIN.INFROM, I, J#1) else MSGSEQ() fi
H9.  I
      in [MAX(SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ)), SOURCE(MS))
          ..MIN(SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ)), SOURCE(MS))]

```

IMP

```

C1.  APORTS#2[J#1].SYSOUT.OUTTO
      sub if SECURE(SEcurity[I], SECURITY[J#1])
        then if APORTS[I].SYSIN.INFROM
              @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
              = MSGSEQ() then MSGSEQ()
              else if J#1
                    = DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))

```

```

and I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))
then  MAIL(APORTS[I].SYSIN.INFROM, I, J#1)
      @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
      else MAIL(APORTS[I].SYSIN.INFROM, I, J#1) fi fi
else MSGSEQ() fi

```

Prover->

.

Prover-> Print Theorem

```

H1.  I
    in [MAX(SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ)), SOURCE(MS))
        ..MIN(SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ)), SOURCE(MS))]
H2. if SECURE(SECURITY[I], SECURITY[J#1])
    then  APORTS[J#1].SYSOUT.OUTTO
          sub MAIL(APORTS[I].SYSIN.INFROM, I, J#1)
          else APORTS[J#1].SYSOUT.OUTTO = MSGSEQ() fi
H3. SECURE(SECURITY[I], SECURITY[J#1])
H4. APORTS[J#1].SYSOUT.OUTTO = APORTS#1[J#1].SYSOUT.OUTTO
H5. APORTS#1[J#1].SYSOUT.OUTTO @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
    = APORTS#2[J#1].SYSOUT.OUTTO
IMP
C1.  APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
    = MSGSEQ()
    -> APORTS#2[J#1].SYSOUT.OUTTO = MSGSEQ()
C2.  (not  APORTS[I].SYSIN.INFROM
      @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
      = MSGSEQ())
    ->  APORTS#2[J#1].SYSOUT.OUTTO
        sub  MAIL(APORTS[I].SYSIN.INFROM, I, J#1)
        @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))

```

The proof of this theorem requires three additional lemmas.

Prover-> Use Lemma

Enter lemma ...

* all A:MsgSeq, all M:Msg, A @ MsgSeq(M) ne MsgSeq();

Lemma added ... Its name is LEMMA#4

.

Prover-> Use Lemma

Enter lemma . . .

* all A,B,C:MsgSeq. A sub B -> A@C sub B@C;

Lemma added . . . Its name is LEMMA#5

Prover-> Use Lemma#3

SWITCHER#4 proved in theorem prover.

Suggest proving VC called SWITCHER#5-> §

Entering Prover with verification condition SWITCHER#5

```

H1: all I#2 : INTEGER [1..100],    APORTS[I#2].SYSIN.OUTTO
      = APORTS#1[I#2].SYSIN.OUTTO
      and APORTS[I#2].SYSOUT.INFROM
      = APORTS#1[I#2].SYSOUT.INFROM
      and APORTS[I#2].SYSOUT.OUTTO
      = APORTS#1[I#2].SYSOUT.OUTTO
      and ( I ne I#2
      -> APORTS[I#2].SYSIN.INFROM
      = APORTS#1[I#2].SYSIN.INFROM)

H2: all J#1 : USERID,    (I ne J#1 -> APORTS[J#1].SYSIN.INFROM = MSGSEQ(I))
      and APORTS[J#1].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SEcurity[J#1],
      MAIL(APORTS[I].SYSIN.INFROM, I, J#1))

H3: APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
      = APORTS#1[I].SYSIN.INFROM

H4: I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))
H5: I = SOURCE(MS)
H6: SECURE(SEcurity[I],
      SEcurity[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))])

```

-->

```

C1: all J#1 : USERID,    (I ne J#1 -> APORTS#1[J#1].SYSIN.INFROM = MSGSEQ(I))
      and APORTS#1[J#1].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SEcurity[J#1],
      MAIL(APORTS#1[I].SYSIN.INFROM, I,

```

J#1))

The scenario for this proof is the same as for Switcher#2. The only additional property required in the proof is Lemma#3.

.

.

.

SWITCHER#5 proved in theorem prover.

Suggest proving VC called SWITCHER#6-> §

Entering Prover with verification condition SWITCHER#6

```

H1: all I#2 : INTEGER [1..100],    APORTS[I#2].SYSIN.OUTTO
      = APORTS#1[I#2].SYSIN.OUTTO
      and APORTS[I#2].SYSOUT.INFROM
      = APORTS#1[I#2].SYSOUT.INFROM
      and APORTS[I#2].SYSOUT.OUTTO
      = APORTS#1[I#2].SYSOUT.OUTTO
      and ( I ne I#2
            -> APORTS[I#2].SYSIN.INFROM
              = APORTS#1[I#2].SYSIN.INFROM)
H2: all J#1 : USERID,  (I ne J#1 -> APORTS[J#1].SYSIN.INFROM = MSGSEQ())
      and APORTS[J#1].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SEcurity[J#1],
        MAIL(APORTS[I].SYSIN.INFROM, I, J#1))
H3: APORTS[I].SYSIN.INFROM @ MSGSEQ(FIRST(APORTS#1[I].SYSIN.BUFQ))
      = APORTS#1[I].SYSIN.INFROM
H4: I = SOURCE(FIRST(APORTS#1[I].SYSIN.BUFQ))
H5: I = SOURCE(MS)
H6: (not SECURE(SEcurity[I],
      SEcurity[DESTINATION(FIRST(APORTS#1[I].SYSIN.BUFQ))]))
-->
C1: all J#1 : USERID,  (I ne J#1 -> APORTS#1[J#1].SYSIN.INFROM = MSGSEQ())
      and APORTS#1[J#1].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SEcurity[J#1],
        MAIL(APORTS#1[I].SYSIN.INFROM, I,
          J#1))

```

This proof is also done like Switcher#2 and uses Lemma#3.

.

.

SWITCHER=6 proved in theorem prover.

Suggest fully defining SECURE-> Print Status All

The current design and verification status is:

Waiting for pending body to be filled in: DESTINATION, MAKEMSG, SECURE,
SOURCE

Proved: NETWORK, SWITCHER

For specifications only: ARRAYTOBAG, ALLSECUREMAILFOR, ISSUBMERGE,
INFROM\ARRAYTOBAG, MAIL, OUTTO\ARRAYTOBAG, SECUREMAIL

Constants/Types: BAGMSGSEQ, MSG, MSGSEQ, MSGTEXT, MSGINBUF, MSGOUTBUF,
MSGBUFARRAY, MSGSEQARRAY, NUSERS, SECURITYARRAY, SECURITYCLASS,
USERID, USERPORT, USERPORTARRAY

Suggest fully defining SECURE-> Explain

<- What are the effects of defining the body of Secure?

No effects.

<- Done

Suggest fully defining SECURE-> Edit

Exec-> Translate Msg.Net

```
function Secure(s1,s2:SecurityClass):boolean =  
  begin  
    exit Secure(s1,s2) iff s1 le s2;  
    Result:=s1 le s2;  
  end;
```

```

type Msg<Source, Destination, Text, MakeMsg, MsgEq> =
  record ( SourcePart, DestPart : UserId;
           TextPart : MsgText
         );
type MsgText = array([1..1024]) of character;

function Source(M:Msg):UserId =
  begin
    exit Source(M) in [1..NUsers] otherwise routineerror;
    cexit Source(M) = M.SourcePart;
    Result := M.SourcePart;
  end;
function Destination(M:Msg):UserId =
  begin
    exit Destination(M) in [1..NUsers] otherwise routineerror;
    cexit Destination(M) = M.DestPart;
    Result := M.DestPart;
  end;
function Text(M:Msg):MsgText =
  begin
    cexit Text(M) = M.TextPart;
    Result := M.TextPart;
  end;
function MakeMsg(S,D : UserId; T : MsgText) : Msg =
  begin
    entry S in [1..NUsers] and D in [1..NUsers];
    cexit MakeMsg(S,D,T) = Msg(S,D,T);
    Result := Msg(S,D,T);
  end;

function MsgEq(M1,M2: Msg): boolean =
  begin
    cexit MsgEq(M1,M2)
      iff M1.SourcePart = M2.SourcePart
        and M1.DestPart = M2.DestPart
        and M1.TextPart = M2.TextPart;

    Result:=M1.SourcePart = M2.SourcePart and
      M1.DestPart = M2.DestPart and
      M1.TextPart = M2.TextPart;
  end;

```


No syntax errors detected.
No semantic errors detected.

This implements messages and the functions which manipulate them.

Exec-> Print Status All

The current design and verification status is:

Waiting for VC generation: DESTINATION, MSGEQ, MAKEMSG, SECURE, SOURCE,
TEXT

Proved: NETWORK, SWITCHER

For specifications only: ARRAYTOBAG, ALLSECUREMAILFOR, ISSUBMERGE,
INFROM\ARRAYTOBAG, MAIL, OUTTO\ARRAYTOBAG, SECUREMAIL

Constants/Types: BAGMSGSEQ, MSG, MSGSEQ, MSGTEXT, MSGINBUF, MSGOUTBUF,
MSGBUFARRAY, MSGSEQARRAY, NUSERS, SECURITYARRAY, SECURITYCLASS,
USERID, USERPORT, USERPORTARRAY

Exec-> Suggest

The suggestion mechanism carries the user through the remainder of the verification. Every proof is automatic and requires no additional information.

Suggest generating VCs for SECURE-> §

Generating VCs for FUNCTION SECURE

Found 1-ST path

Beginning new path...

Assume (unit entry condition)
TRUE

RESULT := S1 le S2

Evaluating SECURE(S1, S2)
Continuing in path...

Leaving unit SECURE
ASSERT RESULT iff S1 is S2

Must verify (unit exit) condition
VC is trivially TRUE

End of path

Suggest generating VCs for SOURCE-> 1

Generating VCs for FUNCTION SOURCE

Found 1-ST path

Relate Concrete and Abstract Specs...

Assume (SOURCE Concrete exit)
SOURCE(M) = M.SOURCEPART

Must verify (SOURCE Abstract exit) condition
Verification condition SOURCE=1
H1: SOURCE(M) = M.SOURCEPART
-->
C1: TRUE

Beginning new path...

Assume (unit entry condition)
TRUE

RESULT := M.SOURCEPART

Evaluating SOURCE(M)

Continuing in path...

Leaving unit SOURCE

ASSERT RESULT = M.SOURCEPART

Must verify (unit exit) condition

VC is trivially TRUE

End of path

Suggest proving VC called SOURCE#1-> §

Entering Prover with verification condition SOURCE#1

H1: SOURCE(M) = M.SOURCEPART

-->

C1: TRUE

Backup point

(.)

Prover-> §Proceeding

SOURCE#1 proved in theorem prover.

Suggest generating VCs for DESTINATION-> §

Generating VCs for FUNCTION DESTINATION

Found 1-ST path

Relate Concrete and Abstract Specs...

Assume (DESTINATION Concrete exit)

DESTINATION(M) = M.DESTPART

Must verify (DESTINATION Abstract exit) condition

Verification condition DESTINATION#1

H1: DESTINATION(M) = M.DESTPART

-->

C1: TRUE

Beginning new path...

Assume (unit entry condition)

TRUE

RESULT := M.DESTPART

Evaluating DESTINATION(M)

Continuing in path...

Leaving unit DESTINATION

ASSERT RESULT = M.DESTPART

Must verify (unit exit) condition

VC is trivially TRUE

End of path

Suggest proving VC called DESTINATION#1-> §

Entering Prover with verification condition DESTINATION#1

H1: DESTINATION(M) = M.DESTPART

-->

C1: TRUE

Backup point

(.)

Prover-> §Proceeding

DESTINATION#1 proved in theorem prover.

Suggest generating VCs for TEXT-> §

Generating VCs for FUNCTION TEXT

Found 1-ST path

Relate Concrete and Abstract Specs...

Assume (TEXT Concrete exit)
TEXT(M) = M.TEXTPART

Must verify (TEXT Abstract exit) condition
Verification condition TEXT#1
H1: TEXT(M) = M.TEXTPART
-->
C1: TRUE

Beginning new path...

Assume (unit entry condition)
TRUE

RESULT := M.TEXTPART
Leaving unit TEXT
ASSERT RESULT = M.TEXTPART

Must verify (unit exit) condition
VC is trivially TRUE

End of path

Page 172

Suggest proving VC called TEXT#1 -> 1
Entering Prover with verification condition TEXT#1

H1: TEXT(M) = M.TEXTPART

-->

C1: TRUE

Backup point

(.)

Prover-> Proceeding

TEXT#1 proved in theorem prover.

Suggest generating VCs for MAKEMSG-> 1

Generating VCs for FUNCTION MAKEMSG

Found 1-ST path

Relate Concrete and Abstract Specs...

Assume (MAKEMSG Concrete exit)

MAKEMSG(S, D, T) = MSG(S, D, T)

Must verify (MAKEMSG Abstract exit) condition

Verification condition MAKEMSG#1

H1: MAKEMSG(S, D, T) = MSG(S, D, T)

-->

C1: TRUE

Beginning new path...

Assume (unit entry condition)
 S in [1..NUSERS] and D in [1..NUSERS]

RESULT := MSG(S, D, T)

Leaving unit MAKEMSG

ASSERT RESULT = MSG(S, D, T)

 Must verify (unit exit) condition

VC is trivially TRUE

 End of path

Suggest proving VC called MAKEMSG#1-> §

Entering Prover with verification condition MAKEMSG#1

H1: MAKEMSG(S, D, T) = MSG(S, D, T)

-->

C1: TRUE

Backup point

(.)

Prover-> §Proceeding

MAKEMSG#1 proved in theorem prover..

Suggest generating VCs for MSGEQ-> §

Generating VCs for FUNCTION MSGEQ

Found 1-ST path

 Relate Concrete and Abstract Specs...

Assume (MSGEQ Concrete exit)

MSGEQ(M1, M2)

iff M1.SOURCEPART = M2.SOURCEPART and M1.DESTPART = M2.DESTPART
 and M1.TEXTPART = M2.TEXTPART

Must verify (MSGEQ Abstract exit) condition

Verification condition MSGEQ#1

H1: MSGEQ(M1, M2)

iff M1.SOURCEPART = M2.SOURCEPART and M1.DESTPART = M2.DESTPART
and M1.TEXTPART = M2.TEXTPART

-->

C1: TRUE

Beginning new path...

Assume (unit entry condition)

TRUE

RESULT := M1.SOURCEPART = M2.SOURCEPART and M1.DESTPART = M2.DESTPART
and M1.TEXTPART = M2.TEXTPART

Leaving unit MSGEQ

ASSERT RESULT

iff M1.SOURCEPART = M2.SOURCEPART and M1.DESTPART = M2.DESTPART
and M1.TEXTPART = M2.TEXTPART

Must verify (unit exit) condition

VC is trivially TRUE

End of path

Suggest proving VC called MSGEQ#1-> §

Entering Prover with verification condition MSGEQ#1

H1: MSGEQ(M1, M2)

iff M1.SOURCEPART = M2.SOURCEPART and M1.DESTPART = M2.DESTPART
and M1.TEXTPART = M2.TEXTPART

-->

C1: TRUE

Backup point

(.)

Prover-> \$Proceeding

MSGSEQ#1 proved in theorem prover.

Suggest one of:

<esc> TRANSLATE (from file)

SAVE (problem on file)

-> Print Status All

The current design and verification status is:

Proved: DESTINATION, MSGEQ, MAKEMSG, NETWORK, SECURE, SOURCE, SWITCHER,
TEXT

For specifications only: ARRAYTOBAG, ALLSECUREMAILFOR, ISSUBMERGE,
INFROM\ARRAYTOBAG, MAIL, OUTTO\ARRAYTOBAG, SECUREMAIL

Constants/Types: BAGMSGSEQ, MSG, MSGSEQ, MSGTEXT, MSGINBUF, MSGOUTBUF,
MSGBUFARRAY, MSGSEQARRAY, NUSERS, SECURITYARRAY, SECURITYCLASS,
USERID, USERPORT, USERPORTARRAY

As indicated by the status summary, the design and verification of the network is complete. The final version of all network programs and their specifications are displayed below.

Suggest one of:

<esc> TRANSLATE (from file)

SAVE (problem on file)

-> Print Unit Alphabetical

Units are printed alphabetically by name.

function ALLSECUREMAILFOR(A : USERPORTARRAY; J : INTEGER; S : SECURITYARRAY)
: MSGSEQARRAY =

begin

exit

(assume all I : USERID, ALLSECUREMAILFOR(A, J, S)[I]
= SECUREMAIL(S[I], S[J],
MAIL(A[I].SYSIN.INFROM, I, J)));

end;

function ARRAYTOBAG(A : MSGSEQARRAY; I, J : USERID) : BAGMSGSEQ =

begin
end;

type BAGMSGSEQ = bag of MSGSEQ;

function DESTINATION(M : MSG) : USERID =
begin
 exit DESTINATION(M) in [1..NUSERS] otherwise ROUTINEERROR;
 coexit DESTINATION(M) = M.DESTPART;
 RESULT := M.DESTPART
end;

function INFROM\ARRAYTOBAG(A : MSGBUFARRAY; I, J : USERID) : BAGMSGSEQ =
begin
end;

function ISSUBMERGE(S : MSGSEQ; A : MSGSEQARRAY) : BOOLEAN =
begin
 exit
 (assume ISSUBMERGE(S, A)
 iff (some X : MSGSEQ, ISMERGE(X, ARRAYTOBAG(A, I, NUSERS))
 and S sub X));
end;

function MAIL(MS : MSGSEQ; I, J : USERID) : MSGSEQ =
begin
 exit
 (assume MAIL(MS, I, J)
 = if MS = MSGSEQ() then MSGSEQ()
 else if SOURCE(LAST(MS)) = I
 and DESTINATION(LAST(MS)) = J
 then MAIL(NONLAST(MS), I, J)
 @ MSGSEQ(LAST(MS))
 else MAIL(NONLAST(MS), I, J) fi fi);
end;

function MAKEMSG(S, D : USERID; T : MSGTEXT) : MSG =
begin
 entry S in [1..NUSERS] and D in [1..NUSERS];
 coexit MAKEMSG(S, D, T) = MSG(S, D, T);
 RESULT := MSG(S, D, T)
end;

```

type MSG <SOURCE, DESTINATION, TEXT, MAKMSG, MSGEQ> =
  record
    (SOURCEPART, DESTPART : USERID;
     TEXTPART          : MSGTEXT);

type MSGBUFARRAY = array (USERID) of MSGOUTBUF;

function MSGEQ(M1, M2 : MSG) : BOOLEAN =
begin
  cexit  MSGEQ(M1, M2)
  iff    M1.SOURCEPART = M2.SOURCEPART and M1.DESTPART = M2.DESTPART
    and M1.TEXTPART = M2.TEXTPART;
  RESULT :=  M1.SOURCEPART = M2.SOURCEPART and M1.DESTPART = M2.DESTPART
    and M1.TEXTPART = M2.TEXTPART
end;

type MSGINBUF = buffer (1) of MSG;

type MSGOUTBUF = buffer (NUSERS) of MSG;

type MSGSEQ = sequence of MSG;

type MSGSEQARRAY = array (USERID) of MSGSEQ;

type MSGTEXT = array ( [1..1024]) of CHARACTER;

process NETWORK(var AUSERPORT : USERPORTARRAY; SECURITY : SECURITYARRAY) =
begin
  block all J : USERID, ISSUBMERGE(AUSERPORT[J].SYSOUT.OUTTO,
    ALLSECUREMAILFOR(AUSERPORT, J, SECURITY));

  cobegin
    SWITCHER(I, AUSERPORT, SECURITY) each I: [1..NUSERS]
  end
end;

const NUSERS = 100;

function OUTTO\ARRAYTOBAG(A : MSGBUFARRAY; I, J : USERID) : BAGMSGSEQ =
begin
end;

function SECURE(S1, S2 : SECURITYCLASS) : BOOLEAN =
begin

```

```

    exit SECURE(S1, S2) iff S1 le S2;
    RESULT := S1 le S2
end;

function SECUREMAIL(S1, S2 : SECURITYCLASS; MS : MSGSEQ) : MSGSEQ =
begin
    exit
        (assume SECUREMAIL(S1, S2, MS)
         = if SECURE(S1, S2) then MS else MSGSEQ() fi);
end;

type SECURITYARRAY = array ( [1..NUSERS]) of SECURITYCLASS;

type SECURITYCLASS = (UNCLASSIFIED, CONFIDENTIAL, SECRET, TOPSECRET);

function SOURCE(M : MSG) : USERID =
begin
    exit SOURCE(M) in [1..NUSERS] otherwise ROUTINEERROR;
    cexit SOURCE(M) = M.SOURCEPART;
    RESULT := M.SOURCEPART
end;

process SWITCHER(I : USERID; var APORTS : USERPORTARRAY;
    SECURITY : SECURITYARRAY) =
begin
    entry I in [1..NUSERS];
    block all J : USERID,
        APORTS[J].SYSOUT.OUTTO
        sub SECUREMAIL(SECURITY[I], SECURITY[J],
            MAIL(APORTS[I].SYSIN.INFROM, I, J))
            and (J ne I -> APORTS[J].SYSIN.INFROM = MSGSEQ());
    var MS : MSG := MAKEMSG(I, I, MSGTEXT());
    keep
        (assume SOURCE(MS) = I);
    loop
        assert all J#1 : USERID,
            APORTS[J#1].SYSOUT.OUTTO
            sub SECUREMAIL(SECURITY[I], SECURITY[J#1],
                MAIL(APORTS[I].SYSIN.INFROM, I,
                    J#1))
            and ( J#1 ne I
                -> APORTS[J#1].SYSIN.INFROM = MSGSEQ());
    begin
        receive MS from APORTS[I].SYSIN;
        if SECURE(SECURITY[I], SECURITY[DESTINATION(MS)])

```



```

    then send MS to APORTS[DESTINATION(MS)].SYSOUT
  end
  when
    is RECEIVEERROR : ;
    is SENDERROR : ;
  end
end
end;

```

```

function TEXT(M : MSG) : MSGTEXT =
begin
  cexit TEXT(M) = M.TEXTPART;
  RESULT := M.TEXTPART
end;

```

```

type USERID = INTEGER [1..NUSERS];

```

```

type USERPORT = record
  (SYSIN : MSGINBUF;
   SYSOUT : MSGOUTBUF);

```

```

type USERPORTARRAY = array (USERID) of USERPORT;

```

Suggest one of:

<esc> TRANSLATE (from file)
 SAVE (problem on file)

-> Print Lemmas All

LEMMA#1 is

```

all A : MSGBUFARRAY,
all B : MSGSEQARRAY,
all X : MSGSEQ,
  ISMERGE(X, OUTTO\ARRAYTOBAG(A, 1, NUSERS))
and (all K : USERID, A[K].OUTTO sub B[K])
-> (some Y : MSGSEQ, ISMERGE(Y, ARRAYTOBAG(B, 1, NUSERS))
    and X sub Y)

```

LEMMA#2 is

```

all A#1 : MSGBUFARRAY,
all I#5 : USERID,
all X#1 : MSGSEQ,

```

```
(all J#1 : USERID, I#5 ne J#1 -> A#1[J#1].INFROM = MSGSEQ(I))
and ISMERGE(X#1, INFROM\ARRAYTOBAG(A#1, I, NUSERS))
-> X#1 = A#1[I#5].INFROM
```

LEMMA#3 is

```
all A, B : MSGSEQ,
all C : MSG,
all K, L : USERID,
all S1, S2 : SECURITYCLASS,      A
                                sub SECUREMAIL(S1, S2,
                                                MAIL(B, K, L))
                                ->  A
                                sub SECUREMAIL(S1, S2,
                                                MAIL(B @ MSGSEQ(C),
                                                K, L))
```

LEMMA#4 is

```
all A#1 : MSGSEQ, all M : MSG, A#1 @ MSGSEQ(M) ne MSGSEQ(I)
```

LEMMA#5 is

```
all A#2, B#1, C#1 : MSGSEQ, A#2 sub B#1 -> A#2@C#1 sub B#1@C#1
```

These are all the lemmas used in the verification of the network.

Suggest one of:

```
<esc> TRANSLATE (from file)
SAVE (problem on file)
```

SID is useful not only during a design and verification, but also afterwards as a maintenance tool. Without a facility like SID, determining the potential effects of changes to a fairly large, complex, and interrelated collection of data is a formidable task. The following is a further illustration of how SID is even more essential as the amount of data grows.

-> Explain

<- What are the effects of changing the header of SecureMail?

Possible effects of changing the header of SECUREMAIL are:

```
Incorrect semantics: ALLSECUREMAILFOR, LEMMA#3, SWITCHER, SECUREMAIL
Invalid verification: NETWORK, SWITCHER
```

<- Why?

Incorrect semantics because calls to SECUREMAIL may become inconsistent. Invalid verification because the external specifications of ALLSECUREMAILFOR, which call SECUREMAIL, were used in proving NETWORK#1. Invalid verification because the external specifications of SWITCHER, which call SECUREMAIL, were used in proving NETWORK#1. Invalid verification because the external specifications of SECUREMAIL, which recursively call SECUREMAIL, were used in proving SWITCHER#4. Invalid verification because basis property LEMMA#3, which calls SECUREMAIL, was used in proving SWITCHER#2, SWITCHER#4, SWITCHER#5, and SWITCHER#6.

<- What are the effects of changing the block specification of Switcher?

Changing the block specification of SWITCHER does not affect SWITCHER if the formula

```
H1: all J : USERID,      APORTS[J].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SEcurity[J],
        MAIL(APORTS[I].SYSIN.INFROM, I, J))
      and (J ne I -> APORTS[J].SYSIN.INFROM = MSGSEQ(I))
```

-->

Changed block specification
is true. If not, some of SWITCHER's VCs are incorrect. The verification of other programs is not affected if the formula

Changed block specification

-->

```
C1: all J : USERID,      APORTS[J].SYSOUT.OUTTO
      sub SECUREMAIL(SEcurity[I], SEcurity[J],
        MAIL(APORTS[I].SYSIN.INFROM, I, J))
      and (J ne I -> APORTS[J].SYSIN.INFROM = MSGSEQ(I))
```

is true. If not, the change invalidates the verification of NETWORK.

<- Why?

The effects on SWITCHER are a change in its VCs. Invalid verification because the block specification of SWITCHER was used in proving NETWORK#1.

<- What are the effects of modifying Lemma#3?

Changing basis property LEMMA#3 does not affect the verification if

Changed lemma

-->

```
CI: all A, B : MSGSEQ,
    all C : MSG,
    all K, L : USERID,
    all S1, S2 : SECURITYCLASS,      A
        sub SECUREMAIL(S1, S2,
            MAIL(B, K, L))
->  A
    sub SECUREMAIL(S1, S2,
        MAIL(B @ MSGSEQ(C),
            K, L))
```

is true. If not, the change invalidates the verification of SWITCHER.

<- Why?

Because it was used in the proof of SWITCHER#2, SWITCHER#4, SWITCHER#5, and SWITCHER#6.

<- Done

Suggest one of:

<esc> TRANSLATE (from file)
SAVE (problem on file)

-> Prove Lemmas#3

*In the above dialog, the user saw that several proofs depend upon Lemmas#3.
So he directs SID to attempt its proof.*

Entering Prover with basis property LEMMA#3

```
all A, B : MSGSEQ,
    all C : MSG,
    all K, L : USERID,
    all S1, S2 : SECURITYCLASS,      A
        sub SECUREMAIL(S1, S2, MAIL(B, K, L))
->  A
    sub SECUREMAIL(S1, S2,
        MAIL(B @ MSGSEQ(C), K,
            L))
```


This proof requires the definitions of Mail, SecureMail, and a new lemma.

LEMMA#3 proved in theorem prover.

Suggest one of:

<esc> TRANSLATE (from file)
SAVE (problem on file)

-> Print Status All

The current design and verification status is:

Proved: DESTINATION, LEMMA#3, MSGEQ, MAKEMSG, NETWORK, SECURE, SOURCE,
SWITCHER, TEXT

For specifications only: ARRAYTOBAG, ALLSECUREMAILFOR, ISSUBMERGE,
INFROM\ARRAYTOBAG, MAIL, OUTTO\ARRAYTOBAG, SECUREMAIL

Constants/Types: BAGMSGSEQ, MSG, MSGSEQ, MSGTEXT, MSGINBUF, MSGOUTBUF,
MSGBUFARRAY, MSGSEQARRAY, NUSERS, SECURITYARRAY, SECURITYCLASS,
USERID, USERPORT, USERPORTARRAY

Suggest one of:

<esc> TRANSLATE (from file)
SAVE (problem on file)

-> Print Lemma Lemma#6

LEMMA#6 is

all $X, Y, Z : \text{MSGSEQ}, X \text{ sub } Y \rightarrow X \text{ sub } Y @ Z$

This is the lemma used in the proof of Lemma#3.

Suggest one of:

<esc> TRANSLATE (from file)
SAVE (problem on file)

-> Save Net.Dmp

That's all, folks!